VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Univerzální GUI využívající malé inteligentní displeje připojené k Raspberry Pi

Universal GUI for a small smart TFT Displays Connected to the Raspberry Pi

2020 Martin Ševčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Ševčík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Univerzální GUI využívající malé inteligentní displeje připojené k Raspberry Pi
Universal GUI for a small smart TFT Displays Connected to the Raspberry Pi

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit GUI ve formě knihovny, umožňující zobrazovat a zadávat parametry aplikací na malých inteligentních displejích připojených k Raspberry Pi.

Student v rámci řešení bakalářské práce zpracuje:
1. Rešerši malých inteligentních displejů a jejich možností.
2. Návrh systému pro GUI.
3. Vytvoření a odladění knihoven pro využití GUI.
4. Vytvoření ukázkové aplikace s použitím navrženého systému GUI.

Seznam doporučené odborné literatury:

[1] NORRIS, Donald. Raspberry Pi: projekty. Brno: Computer Press, 2015. ISBN 9788025143469.
[2] NIXON, Dan. Raspberry Pi Blueprints. Packt Publishing Ltd, 2015. ISBN 9781784392901.
[3] COX, Tim. Raspberry Pi cookbook for Python programmers: over 50 easy-to-comprehend tailor-made recipes to get the most out of the Raspberry Pi and unleash its huge potential using Python. Birmingham : Packt, 2014. ISBN 9781849696623

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **RNDr. Ing. Martin Radvanský, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 11. května 2020 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 11. května 2020 ..... Šerák .....

Abstrakt

Jednou z možných metod, jak může zařízení komunikovat s obsluhou, je zobrazování informací na displeji. Různorodost dostupných displejů a jejich vlastností působí při návrhu problémy s jejich implementací. Tato práce řeší návrh a vytvoření univerzální a snadno rozšiřitelné knihovny, která poskytuje vývojáři zjednodušení při obsluze malých displejů ve vlastní aplikaci. V práci se zaměřuji především na obsluhu inteligentních displejů, které jsou připojeny přes rozhraní UART, I^2C a SPI. Vytvořená knihovna je odladěna na jednočipovém počítači Raspberry Pi. Součástí představeného řešení je také ukázková aplikace ukazující možnosti použitých displejů.

Knihovna má sjednocenou komunikaci dvou značek displejů, 4D Systems a Nextion. Dále pokrývá komunikaci pomocí komunikačního protokolu UART a je připravena k rozšíření komunikace pomocí protokolů I^2C a SPI. Knihovna je navržena tak, aby šla snadno doplnit o další značky displeje a komunikační protokoly. V knihovně lze pomocí jednoduchých metod zapisovat do prvků a získávat z nich hodnoty. Tímto způsobem je zjednodušen celý proces komunikace s displejem. Dále je třeba mít naimplementovanou třídu s požadovaným komunikačním protokolem, třídu s požadovanou značkou displeje, a také k ní vytvořit patřičný konfigurační soubor.

Klíčová slova: chytrý displej; raspberry pi; sériová komunikace; python

Abstract

One way to communicate with the operator is to display information on the display. The variety of available displays and their features causes problems with their implementation during design. This work addresses the design and creation of a versatile and easily extensible library that provides developers with simplicity in handling small displays in their own application. In this work I focus mainly on the operation of intelligent displays, which are connected via UART, I^2C and SPI interfaces. The created library is debugged on a single-chip computer Raspberry Pi. Part of the presented solution is also a sample application showing the capabilities of the used displays.

The library has unified communications between two display brands, 4D Systems and Nextion. It also covers UART communication and is pre-configured to extend I^2C and SPI communication. The library is designed to be easily extended with additional display brands and communication protocols. In the library, you can write elements and into elements and retrieve values using simple methods. This method will simplify the entire communication process with display. Next it is necessary to have a class with the required communication protocol, a class with the required display brand and also to create the appropriate configuration file.

Keywords: smart display; raspberry pi; serial communication; python

Obsah

Seznam použitých zkratek a symbolů

UART	–	Universal Asynchronous Receiver-Transmitter
I^2C (IIC)	–	Inter-Integrated Circuit
SPI	–	Serial Peripheral Interface
MISO	–	Master Input Slave Output
MOSI	–	Master Output Slave Input
IoT	–	Internet of Things
HMI	–	Human Machine Interface
GPIO	–	General Purpose Input/Output
LCD	–	Liquid Crystal Display
TTL	–	Transistor-Transistor Logic
GND	–	Ground
Vcc	–	Voltage at the Common Collector
HW	–	Hardware
Wi-Fi	–	Wireless Fidelity
USB	–	Universal Serial Bus
HDMI	–	High Definition Multimedia Interface
PLC	–	Programmable Logic Controller
SCADA	–	Supervisory Control And Data Acquisition
MES	–	Manufacturing Execution Systems
CNC	–	Computer Numerical Control

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

S rozvojem techniky a rostoucími požadavky lidí se mnoho věci stává automatizovanými, přibývá množství zařízení, se kterými se denně setkáváme a musíme je nějak ovládat. Ať už je to doma, v nemocnicích, v továrnách nebo v kancelářích, všude kolem nás jsou nějaké elektronické zařízení, které se nám jako lidem snaží pomoct. Stále se také u těchto zařízení mění ovládání a snaží se o to, aby bylo co nejjednodušší, nejlevnější a také, aby mělo co nejmenší spotřebu. Trend, který můžeme poslední dobou sledovat, je snaha o vzdálené ovládání přes internet nebo mobilní telefon.

Starší lidé, kteří jsou zvyklí na manuální ovládání, mají problém s tímto typem ovládání, jelikož mnoho z nich tyto technologie příliš dobře neovládá, nebo vůbec nemá přístup k internetu. Pokud by výrobce přidal do zařízení uživatelské rozhraní, podstatně by to zvýšilo jeho náklady a složitost.

Alternativa, která usnadní ovládání i těm, kteří příliš neumí pracovat s internetem, jsou smart displeje. Tyto displeje umožňují vytvořit grafická rozhraní podle vlastní úvahy při minimálních nákladech na vývojáře. Grafická rozhraní se vytváří v programech, které jsou speciálně vytvořené od každého výrobce těchto smart displejů.

Problém však je, že každý výrobce si dělá ovládání podle sebe, jak uzná za vhodné a tím vzniká problém pro programátory. Ti se totiž musí učit od každé značky nejen určitý typ komunikace, (UART, I^2C nebo SPI) ale co je horší, příkazy pro ovládání displejů se mohou značně lišit. Příkladem může být, že některé displeje komunikují pomocí hexadecimálních číslic a jiné pak pomocí textu. Motivací vzniku této práce je ulehčení práce vývojářům hardware pro odstranění problému nejednotné komunikace s displeji.

Cílem této práce je knihovna, která by umožnila programátorovi jednotně naprogramovat funkčnost grafického rozhraní nezávisle na použitém displeji a různým typům komunikace na prototypové platformě Raspberry Pi.

1.1 Zařízení v domácnosti

V domácnosti máme mnoho zařízení, která jsou připojená k elektřině. Všechna tato zařízení zvládají svou základní funkcí, ke které jsou určena, ale často přidávají i mnoho dalších rozšiřujících funkcí. Příkladem takových zařízení, která má většina lidí doma, jsou lednice, myčka na nádobí, trouba, termostat, pračka, kávovar a televize. Dnes je však trendem dělat z těchto zařízení, která mají většinou jednuchou funkci, zařízení o něco chytřejší. Často se těmto zařízením přidává možnost připojení k internetu, čímž můžeme poté zařízení ovládat vzdáleně. Některá zařízení dokaží i data zpracovávat a podle těchto dat i předvídat budoucnost a tak se přizpůsobují našemu životnímu stylu.

1.1.1 Propojená zařízení

Pojmem, který úzce souvisí s naší dobou jsou propojená zařízení. Jedná se o přístroje, které jsou nějakým způsobem propojeny s jinými zařízeními. Nejčastěji jsou tato zařízení propojená přes internet. Mezi propojené zařízení patří i počítače, notebooky a tiskárny, které byly úplně první propojené zařízení v domácnosti.

Dnes se však tento termín používá spíše k označení zařízení, které má možnost se připojit do Internetu a může posílat či přijímat nějaká data. Často se také používá k označení zařízení, jenž lze ovládat vzdáleně, například přes internet.[1]

1.1.2 Smart zařízení

Poslední dobou se lidé stále častěji setkávají se zařízeními označovanými slovem Smart. Smart zařízení jsou rozdílná v tom, že mají i nějakou vlastní naprogramovanou inteligenci. Tato inteligence umožňuje ovládat dané zařízení, podle vnějších podmínek či podnětů, předešlých situací, nebo předpovídat určité události, které mohou nastat.

Se zařízeními smart se váže i složitější hardwarová architektura, než která je u obyčejných propojených zařízení. Architektura Smart zařízení se většinou skládá ze senzorů, mikroprocesorů, datového úložiště, ovládacích prvků a operačního systému. Téměř všechna smart zařízení jsou zároveň propojená zařízení, ale ne každé propojené zařízení je smart zařízení.[1]

1.1.3 Rozdíly

Pro popsání rozdílů mezi smart zařízením a propojeným zařízením si můžeme představit třeba termostat. V případě, že se jedná o obyčejný termostat, musí se přijít k němu a pomocí tlačítek nebo dotykového displeje se nastavit teplota. Zařízení se chová přesně podle svého účelu a nastaví a udržuje požadovanou teplotu v místnosti. V případě, že je termostat propojený, máme další přidané možnosti jej ovládat, nebo se dozvědět teplotu vzdáleně přes síť. To znamená, že umí posílat a přijímat data po připojené síti. Často se také odesílají data výrobcům těchto zařízení k různým statistikám. Smart zařízení však umí ještě něco navíc. Jejich hlavní odlišností od propojeného zařízení je často i umělá inteligence. Smart zařízení se například pomocí strojového učení dokáží naučit denní harmonogram. Zařízení je také schopno zpracovávat dobu návratu domů či dobu vstávání na základě svých senzorů. Toto se může využít poté tak, že hodinu předtím, než dorazí v běžnou dobu domů, nastaví termostat teplotu na požadovanou, ať přijde do vyhřátého domu a zároveň neplýtvá energií tak, že by se teplota vůbec nesnižovala.[2]

1.2 Možnosti řízení

Existuje mnoho možností řízení různých zařízení, ať už třeba v domácnosti, nebo v různých výrobnách. Mezi nejčastější typy řízení patří: řízení pomocí řídicích prvku, řízení pomocí dotykového displeje nebo řízení vzdáleně z externího zařízení.

Displej s ovládacími prvky
Řízení pomocí řídicích prvků a zobrazování pomocí displeje je nejstarším typem řízení z těchto možností. Mezi řídicí prvky mohou patřit klávesnice, myši, dotykové podložky, nebo trackbally. Displej u tohoto typu je většinou pouze pro výstup obrazu.

Dotykový displej
U řízení pomocí dotykového displeje nejsou třeba žádná další zařízení, která slouží k řízení. Stačí pouze displej, který slouží jako vstupní a zároveň i jako výstupní zařízení.

Obyčejný displej s ovládáním ze smartphonu
Při řízení, kde je displej a ovládá se pomocí smartphonu, je potřeba internetové připojení na obou koncích, čili na zařízení a na smartphonu. U zařízení je vidět výstup stavů zařízení a ze smartphonu je možnost tyto stavy pomocí příslušné aplikace měnit.

Bez displeje a s řízením ze smartphonu
U tohoto typu řízení není třeba být u zařízení, aby se zařízení mohlo řídit i se zpětnou vazbou. Zařízení tedy nemá žádné výstupní prvky u zařízení. Pomocí smartphonu je možnost vidět výstupy ze zařízení a zároveň jej z něj řídit.

1.2.1 Výhody a nevýhody

Výhody u řízení pomocí řídicích prvku a zobrazování pomocí jednoduchého displeje jsou, že jsou levné náklady a není potřeba smartphone. Pokud se pokazí zobrazovací zařízení, může se vyměnit nezávisle na řídicích prvcích. U dotykového displeje se také mění pouze displej, ale je dražší, protože je dotykový. Nevýhodou je to, že tato zařízení nelze řídit vzdáleně.

U řízení zařízení ze smartphonů je výhoda, že není nutnost při řízení být v přítomnosti zařízení. U zařízení, které má u sebe displej je výhoda, že si může zjistit stav zařízení i člověk, který nemá u sebe smartphone. Zařízení, které má zobrazovaný stav na smartphone, má výhodu, že se dá zjistit stav i když daný člověk není u zařízení. Nevýhodou těchto zařízení je, že je potřeba internetové připojení na straně zařízení i na straně smartphonu.

1.3 Softwarová podpora displejů

U obyčejných displejů se musí programovat design pomocí vykreslování nějakých obrazců. Například, pro naprogramování tlačítka je potřeba nakreslit vyplněný obdélník podle určitého algoritmu, který si musí programátor naprogramovat sám, nebo musí použít nějakou knihovnu. Dále se musí do tohoto obdélníku vepsat určitý text, také pomocí nějakého algoritmu.

Pokud je displej dotykový, pošle uživateli zprávu, že byl proveden dotyk displeje a souřadnice dotyku. To se může využít k tomu, že se přidá událost, že když se provede dotyk na určité souřad-

nice, tak se provede nějaká funkce. Spojením těchto vlastností se vytvoří základní funkcionalita obyčejného tlačítka.

Tento proces není úplně ideální pro tvoření složitějších obrazovek, protože je zde hodně práce i s tou nejtriviálnější funkcionalitou, jako je tlačítko. Pokud by byla potřeba udělat složitější prvky, byla by doba vytvoření základní funkcionality neúměrně dlouhá.

Komunikace u Smart displejů je zjednodušena díky mikrokontroléru, který zpracovává příkazy, které se posílají na displej z počítače a podle toho odešle odpověď. Tato komunikace je obousměrná, nejen že se dá zobrazovat hodnota na displeji, ale lze i získat zobrazenou hodnotu ukazatele na displeji a displej pošle zpět právě zobrazovanou hodnotu příkazem.

Součástí návrhu GUI pro Smart displeje jsou programy, pomocí kterých je možnost v grafickém editoru uspořádat různé prvky, které jsou potřeba na displeji. V editoru se mohou tedy přesně rozmístit, nastavit se počáteční hodnoty, nebo dokonce si upravit nějakou základní logiku. Poté se z programu vyexportuje soubor, který se následně nahraje do paměti displeje.

1.4 Zaměření a cíle práce

V následujících kapitolách bude provedena reserše dostupných smart displejů. Další část pak tvoří návrh a implementace knihovny pro jejich využití v aplikacích. Poslední částí práce je pak ukázka použití knihovny a návod na její zprovoznění.

2 Teoretická část

V této kapitole bude provedeno seznámení se smart displeji a základními pojmy, na které se může narazit, když se mluví o smart displejích. Dále bude popsán jednodeskový mikropočítač Raspberry Pi, který bude využíván pro komunikaci se smart displeji. Poslední část tvoří rešerše dostupných smart displejů na trhu včetně jejich popisu, vlastností a možností.

2.1 Smart displeje

Smart displeje jsou nazývány smart, protože obsahují integrovanou řídící desku včetně grafického procesoru. Smart displeje mohou být používány bez dalších komponentů k zobrazování, nebo řízení nějakého zařízení. Pokud by měly být všechny komponenty integrovány zvlášť, zabralo by to mnoho času a taky by to stálo mnoho peněz. Používání smart displejů je jednodušší, protože jsou již uzpůsobeny k tomu, že budou sloužit jako komunikační prostředník mezi člověkem a zařízením.[3]

2.1.1 HMI

HMI (Human-Machine Interface) je uživatelské rozhraní nebo přístrojová deska, která propojuje člověka se strojem, systémem nebo zařízením. Zatímco tento pojem může být použit pro každý displej, který umožňuje komunikaci se zařízením, HMI je nejčastěji spojováno v kontextu s průmyslem.

Ačkoliv HMI je nejčastěji pojmem technologie, občas je spojováno s MMI (Man-Machine Interface), OIT (Operator Interface Terminal), LOI (Local Operator Interface) nebo OT (Operator Terminal). HMI a grafická rozhraní mají podobný význam, nejsou však synonymy. Grafická rozhraní jsou často využívána v HMI pro větší přehlednost. V průmyslu se HMI může používat k vizuálnímu zobrazení dat, sledování vyrobního procesu, monitorování strojových vstupů a výstupů, atd.[4]

Podobně jako lidé komunikují v domě s regulátorem teploty, tak komunikují například i operátoři výroby s HMI, aby sledovali a kontrolovali teplotu vody v nějakém chladiči, nebo jestli běží motor.

HMI zařízení jsou v mnoha podobách. Od zabudovaných displejů na strojích, přes počítačové monitory po tablety, ale nehledě na jejich podobu nebo v jakém kontextu se využívají, jejich hlavní účel je poskytovat náhled do strojové části a přetvořit jej do lidsky čitelné podoby.

2.1.2 HMI a PLC

HMI se hodně používají v oblasti PLC systémů (Programmable Logic Controllers) a zobrazují uživatelům jeho vstupy a výstupy. V této oblasti se většinou používají HMI displeje, které mají velikost 5 až 10 palců. HMI a PLC je většinou propojeno pomocí LAN kabelu, nebo pomocí komunikačních protokolů RS-232, RS422, nebo RS-485. HMI displeje mohou být použity pro

jednu funkci, jako třeba monitorování produkce, nebo pro vykonávání složitějších procesů jako třeba zapínání či vypínání určitých částí stroje, nebo zvyšování rychlosti či počtu otáček. Vše záleží na konkrétní implementaci.

HMI se používají pro optimalizaci výrobního procesu tím, že data jsou v digitální podobě a jsou centralizována. Navíc se může na HMI naimplementovat funkce, že se daná data zobrazí graficky a tím se stanou přehlednějšími. Data mohou být vizualizována v podobě grafů či statistik. HMI může mít také nějaké upozornění, když je překročena určitá hodnota výrobního procesu. HMI se může propojit s výrobními systémy SCADA a MES a to vše přes jednu konzoli. Používají se v oblastech výroby energií, výroby či zpracování jídla, výrobních linek, těžení ropy a zemního plynu, recyklace nebo hromadné dopravy.[4]

Kdysi museli operátoři chodit po hale od stroje ke stroji a neustále kontrolovat hodnoty a zapisovat je do nějakého výkazu. Tím, že nastoupily PLC systémy, které komunikují v reálném čase s HMI displejem, se zrušila tato nutnost chodit z jedné části haly do druhé a usnadnila tak práci operátorům. Výhodou je také odstranění lidských chyb, jako je třeba přepsání se v desetinné čárce.

2.1.3 HMI a mikrokontroléry

HMI displeje se také často připojují k mikrokontrolérům. Mikrokontrolér je jednočipový počítač, který je většinou velmi malých rozměrů. Tyto displeje se propojují s mikrokontrolérem většinou pomocí rozhraní UART, I^2C, nebo SPI. Displeje od různých značek mají různá vývojová prostředí, ve kterých se upravuje vzhled a struktura obrazovky, nebo více obrazovek. Každá značka displeje má jiné prvky, které je možno využívat. Mezi tyto prvky patří například různá tlačítka, posuvníky, ukazatele, nebo číselné, či textové výstupy. V grafickém vývojovém prostředí se tedy libovolně nastaví různě prvky, které svým uživatelům určitá značka poskytuje. Ve většině grafických vývojových prostředí je možnost i nastavit základní funkce různých prvků. Například, že když klikneme na tlačítko, tak se změní obrazovka na jinou.

2.2 Rešerše zástupců smart displejů

V této podkapitole jsou popsány různé displeje a jejich výrobci. Značky těchto displejů jsou jedny z nejpopulárnějších. Jedná se o značky 4D Systems, Nextion, MikroElektronika, Electronic Assembly a Riverdi.

2.2.1 4D Systems

Obrázek 1: Produkt značky 4D Systems

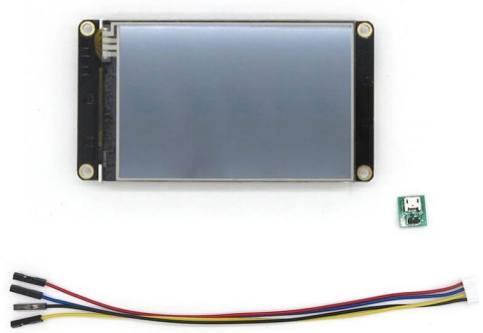
Displeje značky 4D Systems se prodávají se dvěma různými typy displejů. První skupina má typ displeje OLED a druhá skupina displejů má typ LCD. Na obrázku 1 [5] se nachází displej uLCD-32PTU.

Skupina s typem displeje OLED má 10-pinové rozhraní, které slouží pro zapojení různých typů zařízení. Tyto displeje obsahují 10 KB flash paměti, která slouží pro trvalé úložiště displeje a obsahuje 510 bajtů operační paměti. Tyto displeje se prodávají ve velikostech 0,96” s rozlišením 96×64 pixelů, 1,5” s rozlišením 128×128 pixelů a 1,7” s rozlišením 160×128 pixelů.

Skupina s typem displeje LCD o velikostech 2,4”, 2,8” a 3,2” s rozlišením 240×320 pixelů má 5-pinové UART rozhraní pro komunikaci a také obsahuje I^2C rozhraní. Displej má 14 KB flash paměti pro ukládání uživatelského kódu a 14 KB operační paměti SRAM.

Další velikosti ve skupině s typem displeje LCD jsou o velikostech 1,38” s rozlišením 220×220 pixelů, 3,5” s rozlišením 320×480 pixelů, 4,3” s rozlišením 480×272 pixelů, 7,0” s rozlišením 800×480 pixelů a 9,0” s rozlišením 800×480 pixelů. Tyto displeje mají 32 KB operační paměti typu SRAM a 32 KB flash paměti pro kód aplikace a data. Hlavní komunikační rozhraní je přes 5-pinový UART a vedlejší komunikační rozhraní jsou $3\times I^2C$ a 3×SPI.[6] Strukturovaný popis displejů a jejich parametrů se nachází v příloze v tabulce 6.

2.2.2 Nextion



Obrázek 2: Produkt značky Nextion

Displeje značky Nextion se dělí do tří základních skupin. Basic, Enhanced a Intelligent. Rozdíl v těchto skupinách jsou možnosti displeje. Skupina Basic má nejméně dostupných prvků v grafickém editoru, skupina Enhanced má některé prvky v grafickém editoru navíc a skupina Intelligent má všechny, které značka Nextion poskytuje. Tyto skupiny se samozřejmě liší cenou.[7] Na obrázku 2 [8] se nachází displej NX4832K035.

Skupina Basic má displeje ve velikostech 2,4 palců s rozlišením 240 × 320 pixelů, 2,8" s rozlišením 240 × 320 pixelů, 3,2" s rozlišením 400 × 240 pixelů, 3,5" s rozlišením 480 × 320 pixelů, 4,3" s rozlišením 480 × 272 pixelů, 5,0" s rozlišením 800 × 480 pixelů a 7,0" s rozlišením 800 × 480 pixelů. Displeje od velikosti 2,4" do 3,2" mají 4 MB flash paměti pro trvalé úložiště. Ostatní displeje mají flash paměti 16 MB. Všechny tyto displeje mají 3584 bajtů operační paměti a rezistivní dotykovou vrstvu.[9]

Skupina Enhanced má displeje ve stejných velikostech i rozlišeních jako skupina Basic. Avšak je zde přidán další 7,0" displej, který má kapacitní displej. Displeje o velikostech od 2,4" do 3,2" mají 16 MB flash paměti pro trvalé úložiště a 3584 bajtů operační paměti. Ostatní displeje mají 32 MB flash paměti pro trvalé úložiště a 8192 bajtů operační paměti. Všechny displeje mají rezistivní dotykovou vrstvu, až na výše zmiňovaný s kapacitním dotykovým snímačem.[10]

Skupina Intelligent má displeje ve velikostech 7,0" s rozlišením 800 × 480 pixelů a 10,1" s rozlišením 1024 × 600 pixelů, ale typů displejů má šest. Čtyři displeje ve velikosti 7,0" a dva ve velikosti 10,1". 7,0" se prodávají v kombinacích: rezistivní dotyková vrstva a bez rámečku, rezistivní dotyková vrstva s rámečkem, kapacitní snímač bez rámečku a kapacitní snímač s rámečkem. 10,1" displeje se prodávají pouze buď s kapacitním snímačem, nebo s rezistivním dotykovou vrstvou. Všechny tyto displeje mají 128 MB flash paměti pro trvalé úložiště a 512 KB operační paměti.[11] Strukturovaný popis displejů a jejich parametrů se nachází v příloze v tabulce 7.

2.2.3 MikroElektronika

Obrázek 3: Produkt značky MikroElektronika

Displej s uhlopříčkou 3,5” a rozlišením 320 × 240 pixelů. HMI obsahuje 256 KB operační paměti flash a 8 MB pro úložiště dat. Existuje ve třech verzích. Jedna verze je bez snímače dotyku, druhá verze má rezistivní dotykovou vrstvu a třetí verze má kapacitní snímač.[12] Na obrázku 3 [13] se nachází displej HMI 3,5”.

Displej s uhlopříčkou 4,3” má rozlišením 480 × 272 pixelů, displej s uhlopříčkou 5,0” má rozlišením 800 × 480 pixelů a displej s uhlopříčkou 7,0” má rozlišením 800 × 480 pixelů. HMI obsahuje 256 KB operační paměti flash a 8 MB pro úložiště dat. Existuje ve čtyřech verzích. Jedna verze je bez snímače dotyku, druhá verze má rezistivní dotykovou vrstvu, třetí verze má kapacitní dotykový snímač a čtvrtá verze má také kapacitní dotykový snímač a navíc má dekorační rámeček okolo displeje.[14][15][16] Strukturovaný popis displejů a jejich parametrů se nachází v příloze v tabulce 8.

2.2.4 Electronic Assembly

Obrázek 4: Produkt značky Electronic Assembly

Electronic Assembly je firma, která poskytuje smart displeje o čtyřech velikostech. 2,8” s rozlišením 128×64 pixelů, 3,2” s rozlišením 160×104 pixelů, 4,2” s rozlišením 240×128 pixelů a 5,7” s rozlišením 320×240 pixelů. Každou velikost tohoto displeje lze zakoupit ve čtyřech variantách. První varianta je s modrým pozadím a bez dotykového displeje, druhá varianta je s modrým pozadím a rezistivní dotykovou vrstvou, třetí varianta je s bílým pozadím a bez dotykového displeje a čtvrtá varianta je s bílým pozadím a rezistivní dotykovou vrstvou. Displeje o velikostech 2,8”, 3,2” a 4,2” mají 64 KB flash paměti pro trvalé úložiště a displej o velikosti 5,7” má 80 KB flash paměti pro trvalé úložiště. Všechny displeje mohou komunikovat pomocí třech komunikačních protokolů. Komunikačními protokoly jsou RS-232, I^2C a SPI.[17][18][19][20] Strukturovaný popis displejů a jejich parametrů se nachází v příloze v tabulce 9. Na obrázku 4 [17] se nachází displej EA eDIP128W-6LWTP.

2.2.5 Riverdi

Obrázek 5: Produkt značky Riverdi

Značka Riverdi má mnoho smart displejů. Velikosti displejů jsou 2,83”, 3,5”, 4,3”, 5,0” a 7,0”. Displeje o velikosti 2,83” a 3,5” mají rozlišení 240×320 pixelů. Displeje o velikosti 4,3” mají rozlišení 480×272 pixelů a displeje o velikosti 5,0” a 7,0” mají rozlišení 800×480 pixelů. Varianty typu displejů jsou bez dotyku, s kapacitním snímačem a s rezistivní vrstvou. Varianty komunikačního rozhraní jsou I^2C a SPI. Většina z těchto displejů má mnoho variant, takže jsou všechny kombinace velikostí, typu dotykového displeje a různých komunikačních rozhraní.[21] Strukturovaný popis displejů a jejich parametrů se nachází v příloze v tabulce 10. Na obrázku 5 [22] se nachází displej RiTFT-35-CAP.

2.3 Platforma Raspberry Pi

Raspberry Pi je malý počítač, který se skládá pouze ze základové desky, která obsahuje všechny nutné obvody k základnímu běhu počítače. Základová deska Raspberry Pi je asi o velikosti platební karty. Na Raspberry Pi je možnost mít různé aplikace, které jsou běžné na desktopových

počítačích. Mohou to být například textové či tabulkové editory, různé ovladače osvětlení, jednoduché servery a nějaké jednodušší hry. Je to zkrátka takový zmenšený počítač, který však má podstatně slabší hardware oproti desktopovým počítačům.

2.3.1 Hardware

Procesor

V první generaci Raspberry Pi byl používán 700 MHz ARM1176JZF-S procesor. Tento procesor měl L1 Cache o velikosti 16 KB a L2 Cache o velikosti 128 KB. L2 Cache byla hlavně využívána grafickým procesorem VideoCore IV.

Dřívější model Raspberry Pi 2 využíval 900 MHz 32-bitový čtyřjádrový procesor ARM Cortex-A7 s 256 KB sdílené L2 Cache. Pozdější verze Raspberry Pi 2 byla upgradovaná na 1,2 GHz 64-bitový čtyřjádrový procesor ARM Cortex-A53.

Raspberry Pi 3 Model B používá 1,2 GHz 64-bitový čtyřjádrový procesor ARM Cortex-A53, což je stejný typ jako u pozdější verze Raspberry Pi 2. Pozdější modely Raspberry Pi 3 Model A+ a Model B+ mají frekvenci procesoru 1,4 GHz.

Raspberry Pi 4 Model B má v sobě 1,5 GHz 64-bitový čtyřjádrový procesor typu ARM Cortex-A72. Obsahuje 1 MB sdílené L2 Cache.

Raspberry Pi Zero, což je redukovaná verze v počtech vstupních a výstupních pinů, má v sobě stejný procesor, jako měla první generace Raspberry Pi, což je ARM1176JZF-S. Nicméně tento procesor je nataktovaný na 1 GHz místo 700 MHz.[23]

RAM

První designy Raspberry Pi Model A a B obsahovaly pouze 256 MB operační paměti. Tyto designy měly možnost rozdělit RAM třemi způsoby. Výchozí způsob rozdělení byl 192 MB pro procesor a 64 MB pro GPU, což by mělo stačit pro samostatné dekódování videa o rozlišení 1080p, nebo jednoduchou práci s 3D. Druhý způsob rozdělení byl 224 MB pro procesor a 32 MB pro GPU. Tento způsob rozdělení se používal pro obyčejnou práci na linuxu, kde by nejspíš jakékoliv zpracování videa nebo 3D selhalo. Třetí způsob rozdělení se používá pro náročnější práci s videem a jeho dekódováním. Tento způsob rozděluje paměť mezi CPU a GPU rovnoměrně, což znamená 128 MB pro procesor a 128 MB pro grafiku.

Pozdější model Raspberry Pi Model B měl 512 MB RAM. U tohoto modelu bylo dělení opět třemi způsoby. 256 MB pro procesor a 256 MB pro grafiku, 384 MB pro procesor a 128 MB pro grafiku a 496 MB pro procesor a 16 MB pro grafiku.

O chvíli později se přidala možnost si zvolit rozdělení mezi procesor a grafiku podle sebe. Toto se dělalo pomocí zápisu do souboru, kde se určilo, kolik MB přiřadíme GPU. Možnost byla od 16 MB do 256 MB po 8MB krocích. Tento způsob rozdělení bylo možné použít na starší i novější typ Raspberry Pi.

Raspberry Pi 2 Model A i Model B mají 1 GB RAM.

Raspberry Pi 3 Model A+ má méně než Raspberry Pi 2, a to 512 MB. Model B a Model B+ má stejně jako Raspberry Pi 2, což je 1 GB.

Raspberry Pi 4 má více variant. Výběr je z velikostí 1 GB, 2 GB, 4 GB a 8 GB. Samozřejmě, čím větší velikost RAM, tím lepší výkon, ale také větší cena.[23]

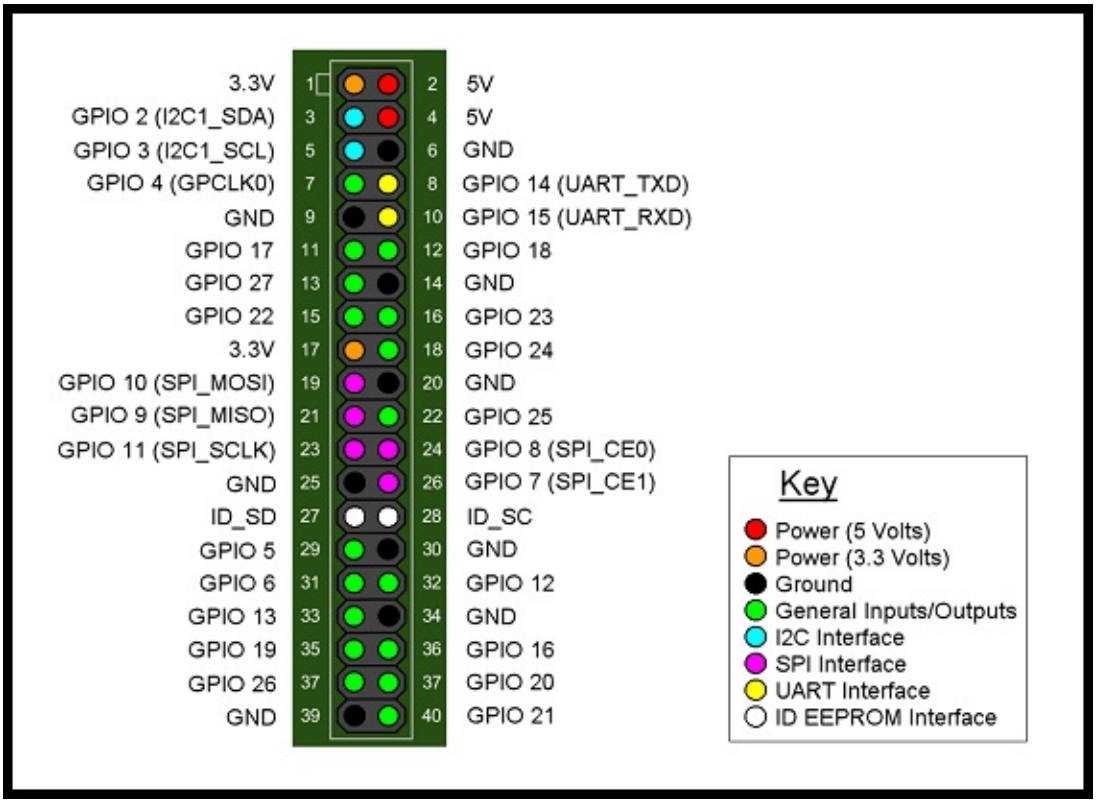
Připojení k síti

U Raspberry Pi modelů A a A+, není připájen konektor pro ethernetový kabel. Pokud se uživatel chce připojit k internetu na těchto modelech, je třeba použít bezdrátový Wi-Fi modem, nebo redukci z USB na Ethernet. Na modelech B a B+ se už nachází Ethernetový port, tím pádem stačí připojit jen Ethernetový kabel.

Raspberry Pi 3 a Raspberry Pi 4 mají už přímo na destičce Wi-Fi modem, takže se odstraňuje problém s kompatibilitou, kde některé modemy defaultně nespolupracovaly s Raspberry Pi a musely se dohledávat ovladače a manuálně konfigurovat.[23]

GPIO piny

GPIO piny (z angličtiny General purpose input-output pins), jsou piny, které je možno naprogramovat k vlastní funkčnosti. Pořadí pinů je vyobrazeno na obrázku 6. [24] Piny mohou být buď vstupní nebo výstupní. Vstupní piny mohou mít k sobě připojené například nějaké tlačítko a výstupní například LEDku. To znamená, že tento malý počítač může být využíván jako prostředek ke zpracování různých fyzikálních veličin, když se připojí na GPIO piny potřebný senzor.[25]



Obrázek 6: Rozložení GPIO pinů

Ne všechny piny jdou naprogramovat, protože některé piny se používají jako napájení nebo zemnění. Z celkových 40 pinů se dá použít jako GPIO 26 pinů. Zbývá tedy 8 pinů pro zemnění, 2 piny pro napájení 3,3 V, 2 piny pro napájení 5 V a 2 piny ID EEPROM.[26]

Některé GPIO piny mají alternativní funkci pro práci s různými komunikačními protokoly. Avšak i tyto piny se dají přeprogramovat pro vlastní účely. Mezi ně patří:

- I^2C (GPIO 2, GPIO 3)
- UART (GPIO 14, GPIO 15)
- SPI (GPIO 7, GPIO 8, GPIO 9, GPIO 10, GPIO 11)[26]

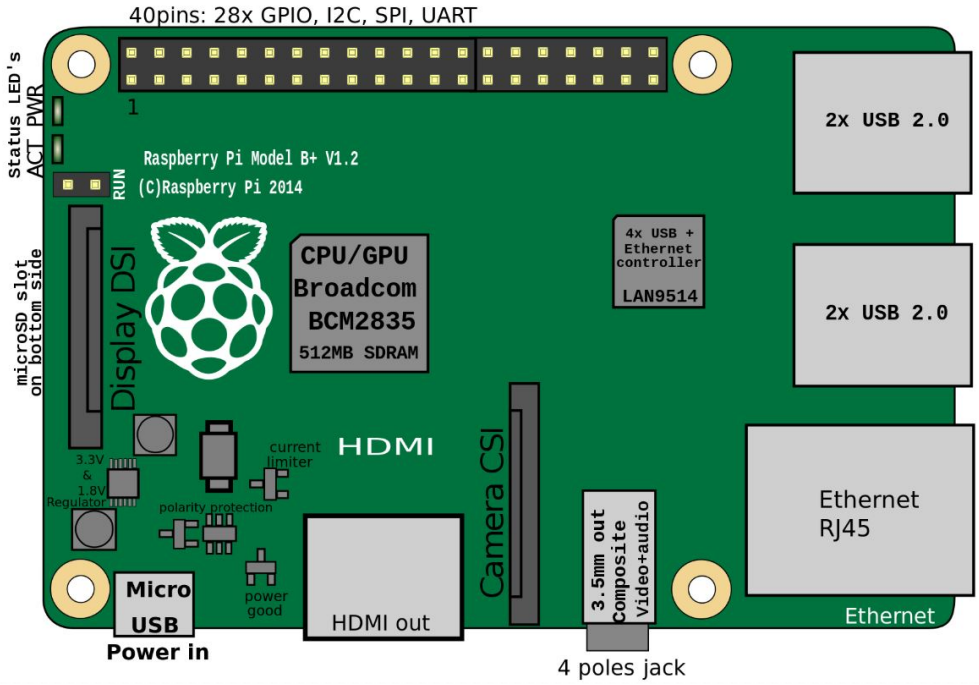
Na pinech s označením 2 a 4 se nachází 5V napětí a na pinech 1 a 17 se nachází 3,3V napětí, které je možno použít podle toho, jaké napětí daná součástka vyžaduje. Dále je potřeba samozřejmě propojit toto napětí s jedním ze zemnících pinů, což jsou piny s označením 6, 9, 14, 20, 25, 30, 34 a 39, aby se uzavřel okruh.[25]

Piny s číslem 27 a 28 s označením ID EEPROM se nepoužívají pro klasické zapojení kabelů. Tyto dva piny jsou využity pro správnou funkčnost pokud připojíme HAT (Hardware Attached on Top) destičku na Raspberry Pi, která rozšiřuje funkčnost Raspberry Pi. Na trhu je těchto destiček spousta a umožňují například zautomatizovat dům, připojit elektronické klávesy, nebo

získávat data ze senzorů přímo na destičce, jako je teplota, tlak, vlhkost nebo celková orientace.[25]

Architektura Raspberry Pi 2 Model B

Celé Raspberry Pi 2 Model B ovládá CPU/GPU Broadcom BCM2835 s 1024 MB operační paměti. Napájení je vyřešeno pomocí MicroUSB portu. Pokud je potřeba obrazový výstup, je možnost si připojit monitor pomocí HDMI portu, nebo DSI portu. Pro připojení kamery lze použít CSI port. Obsahuje 4× USB verze 2.0, která slouží pro připojení periférií. Dále je možnost připojit periférie pomocí 40 pinů. Raspberry Pi 2 Model B umožňuje také připojit mikrofón, či sluchátka pomocí 3,5mm jacku. Pro ukládání na trvalé úložiště a bootování systému slouží MicroSD slot, kde je potřeba připojit MicroSD kartu.[23] Kde a jak jsou prvky napájeny na destičce Raspberry Pi je vyobrazeno na obrázku 7. [27]



Obrázek 7: Architektura Raspberry Pi 2 Model B

2.3.2 Podpora operečních systémů v Raspberry Pi

Nejčastější volbou operačního systému pro Raspberry Pi je Raspbian. Tento operační systém dokáže využít platformu Raspberry Pi na maximum, protože je vytvářený přímo pro obecné potřeby uživatelů Raspberry Pi. Na Raspberry Pi lze však nainstalovat mnoho dalších operačních systémů. Často jsou tyto operační systémy variací linuxu, například OSMC, OpenELEC, Lakka, RaspBSD, RetroPie, Ubuntu Core nebo Linutop. Mezi další operační systémy, které nejsou lixnovou distribucí jsou RISC OS a Windows IoT Core. Pokud se zvolí operační systém, který

není dělaný přímo pro platformu Raspberry Pi, tak hrozí, že mnoho věcí se bude dělat složitějším způsobem, než jak by tomu bylo u Raspbianu.[28]

2.3.3 Využití Raspberry Pi v projektu

Díky velké rozšířenosti této platformy a neustále se vyvíjecím novým verzím, je tato platforma, která byla původně využívána spíš jako zábava pro programátory, často používána i v průmyslové praxi. Vývojář si může také zvolit verzi Raspberry Pi téměř podle svých výkonnostních, velikostních, nebo funkcionálních potřeb, protože verzí je mnoho. Cenová politika je také velmi přijatelná a existuje obrovská komunita, kde lze získat informace a podporu. Proto padlo rozhodnutí ladit knihovny především na této platformě, ale nic nebrání, díky Pythonu, využít i nějakou alternativní hardwarovou platformu, jako je například Radxa, RockPi, a tak dalé.

2.4 Sériová komunikace

Sériové komunikaci se rozumí komunikace, kde se posílá pouze jeden bit v daném okamžiku. Narozdíl od paralelní komunikace, kde se posílá více bitů najednou, které jsou připojeny na více datových kanálů. Data se posílají přes komunikační kanál nebo přes počítačovou sběrnici. Vestavěná elektronika a její vnitřní obvody, jako je například procesor, si musí nějakým způsobem vyměňovat informace. Tato komunikace musí být postavena na stejném protokolu vysílače a přijímače. V komunikačních sítích se již definovalo mnoho protokolů, ale všechny mohou být zařazeny do dvou skupin. Těmi skupinami jsou paralelní komunikace a sériová komunikace.

2.4.1 Asynchronní sériová linka

Existují různé varianty sériových rozhraní, které byly vytvořeny, aby splnily různé požadavky vestavěných systémů. USB a Ethernet jsou dvě nejznámější sériové rozhraní. Ostatní, velmi běžné sériové rozhraní, jsou SPI a I^2C sběrnice. Všechny sériové rozhraní mohou být přiřazené do jedné ze dvou skupin: synchronní a asynchronní.

Synchronní sériové rozhraní vždy ke své činnosti využívá hodinový signál, takže všechna zařízení na synchronní sériové sběrnici sdílejí stejný hodinový signál. Rychlejší přenos je zajištěn tím, že mohou být vynechány start a stop bity a mezera mezi jednotlivými bajty, takže synchronizace je pomocí vzestupné nebo sestupné hrany hodinového signálu. Příklad synchroního rozhraní je SPI a I^2C sběrnice.

Asynchronní znamená, že data jsou přenášena bez pomoci vnějšího hodinového signálu a synchronizace je prováděna pomocí start a stop bitů. Tato přenosová metoda je ideální pro minimalizování počtu vodičů a pinů pro vstup a výstup, ale to taky znamená že potřebujeme nějakým způsobem zajistit spolehlivý přenos dat.[29]

2.4.2 Parametry sériového přenosu

Asynchronní sériový protokol má mnoho zabudovaných pravidel. Tato pravidla pomáhají zajistit robustní a bezchybný přenos dat. Uvedené mechanismy dostaneme, pokud odstraníme vnější hodinový signál:

- datové bity
- synchronizační bity
- paritní bity
- přenosová rychlost [29]

Přenosová rychlost

Přenosová rychlost určuje, jak rychle budou data přenášena přes sériové rozhraní. Tato rychlost je obvykle vyjádřena v bitech za sekundu (anglicky bits per second - bps). Převrácená hodnota přenosové rychlosti udává čas potřebný k přenesení jednoho bitu. Tato hodnota určuje, jak dlouho vysílač drží sériovou linku ve vysoké či nízké logické úrovni, nebo v jakém intervalu přijímací zařízení vzorkuje svou linku.

Přenosová rychlost může být nastavena na téměř jakoukoliv, nicméně závisí na délce vedení, čím delší je vedení, tím je menší přenosová rychlost. Také prostředí a rušení může mít vliv na nižší přenosovou rychlost. Důležitý požadavek na přenosovou rychlost je, že obě koncová zařízení přenáší data stejnou rychlostí. Jednou z nejběžnější přenosovou rychlostí, kde rychlost není kritická, je 9600 bps. Další běžné rychlosti jsou 1200, 2400, 4800, 19200, 38400, 57600 a 115200 bps.

Čím vyšší je přenosová rychlost, tím rychleji jsou data přenášena, nicméně limit je dán výkonem mikrokontroléru. Rychlost většinou nepřekračuje hodnotu 115200 bps, což je hodně pro některé mikrokontroléry. Když je zvolena příliš vysoká rychlost, tak se začnou objevovat chyby na části přijímače, protože vstupní obvody nemusí vyšší rychlosti stíhat.[29]

Přenášená data

Každý blok přenesených dat je poslán v paketu nebo rámci bitů. Rámce jsou vytvořeny tím, že se přidají synchronizační a paritní bity. Důležitá část každého sériového paketu jsou data, které přenáší. Tato část se nazývá částí, protože její velikost není předem určená. Množství dat v každém paketu může být nastaveno od 5 do 9 bitů. Nejběžnější množství dat je 8 bitů, což je 1 bajt. Jiné délky se používají z důvodu efektivity, například, když se přenáší ascii znaky, stačí přenést pouze 7 bitů.

Po souhlasu s délkou paketu, se musí obě zařízení shodnout na tom, zda budou používat little-endian nebo big-endian. Zda nejvýznamnější bit bude poslán jako první nebo jako poslední. Pokud to není jinak určeno, počítá se s little-endianem, což znamená, že nejméně významný bit bude na začátku paketu.[29]

Synchronizační bity

Synchronizační bity jsou dva nebo tři speciální bity přeneseny s každou částí dat. Jsou to bity, které určují, že část začíná a končí. Vždy se začíná jedním bitem, ale končí se jedním nebo dvěma. Většinou to bývá jeden.

Počáteční bit je vždy indikovaný nečinnou datovou linkou 1 nebo 0, zatímco stop bit přejde do klidového stavu tím, že bude držet 1.[29]

Paritní bity

Parita je způsob, jak velmi jednoduše a nízkoúrovňově kontrolovat chyby. Parita může být buď sudá, nebo lichá. Pro vytvoření parity jsou všechny datové bity sečteny a sudost, či lichost rozhodne, jestli bude paritní bit nastaven na 1 nebo na 0. Když bude v datech sudý počet jedniček, takže výsledná suma bude sudá, bude parita nastavena na 0. V opačném případě, pokud bude v datech lichý počet jedniček, parita se nastaví na 1.

Parita je volitelná, a není moc rozšířená. Parita může být užitečná, když se data přenáší přes rušné média. Tím, že se bude používat parita se zvýší objem paketu a tím se sníží počet přenesených paketů za minutu. Také musí mít jak vysílací, tak přijímací zařízení implementovaný zpracovávač chyb. Většinou tak, že se požádá o přeposlání paketu, v kterém se vyskytla chyba.[29]

2.4.3 Hardware sériové linky

Sériová sběrnice se skládá pouze ze tří vodičů, jeden pro posílání dat, druhý pro přijímání dat a třetí společný pro uzemnění. Tím pádem by sériové zařízení mělo mít dva sériové piny, přijímač RX a vysílač TX a jeden vodič na uzemnění. Je důležité si uvědomit že vodiče RX a TX z prvního zařízení jsou prohozené v druhém zařízení. To znamená, že vodič RX z jednoho zařízení, musí jít do TX druhého zařízení a naopak. Většina elektroniky se připojuje tak, že jdou stejné kabely ke stejným, ale tady nastává výjimka. V tomto případě dává smysl, že jedním zařízením se data vysílají a druhé zařízení je přijímá.

Sériové rozhraní, kde obě zařízení mohou vysílat a přijímat data je buď full-duplexní nebo half-duplexní. Full-duplex znamená, že obě zařízení mohou vysílat a přijímat zároveň. Half-duplexová komunikace znamená, že sériové zařízení se musí střídat v komunikaci. V jednu chvíli jedno zařízení vysílá, druhé přijímá a nebo naopak.

Některé sériové sběrnice mohou existovat pouze s jedním typem připojení. Například sériově připojené LCD, které pouze naslouchá a neposílá žádné odpovědi. Toto se také někdy nazývá jako simplex. Tím pádem stačí pouze aby byl připojen vodič TX z vysílacího zařízení do RX přijímacího zařízení.[29]

Fyzická vrstva sériové linky

Existují určité standardy implementace sériové linky na úrovni signálů. Asi jako nejznámější hardwarové implementace jsou RS-232 a TTL.

Když komunikují mikrokontroléry a jiné nízkoúrovňové integrované obvody sériově, obvykle komunikují na TTL úrovni. TTL sériový signál existuje mezi 0 V a mikrokontrolérovým napájecím napětím, které je obvykle 3,3 V až 5 V. Signál na úrovni Vcc indikuje binární jedničku a signál na úrovni GND, což je běžně 0 V, indikuje binární nulu. To znamená, že binární jednička nastane v případě počátečního stavu, dat o hodnotě jedna a stopového bitu, zatímco binární nula v případě startovního bitu a dat o hodnotě binární nuly.

RS-232, které může být nalezeno na zastaralejších počítačích a periferiích, je jako opačné TTL. RS-232 signály jsou obvykle mezi -13 V a 13 V, i když specifikace uvádí od ± 3 V do ± 25 V. Na těchto signálech spodní hranice napětí určuje buď nečinnou linku, stopový bit a nebo datový bit o hodnotě jedna. Vrchní hranice napětí zase určuje startovní bit nebo data o hodnotě nula. Proto je na RS-232 vše opačně jako u TTL.

Mezi těmito dvěma standardy sériových signálů je TTL mnohem jednodušší naimplementovat do vestavěného okruhu. Nicméně nižší napěťové úrovně jsou citlivější na ztrátu při přenosu. RS-232, nebo jeho složitější standardy jako RS-485, jsou vhodnější pro sériový přenos na delší vzdálenost.

Když se připojují dvě sériové zařízení dohromady, je důležité si ověřit, zda se jejich standardy shodují. Není možno připojit přímo rozhraní s TTL standardem a RS-232 standardem. Aby byla zajištěna funkčnost, musely by se tyto signály upravit.[29]

2.4.4 Příklady využití

Sériové porty se stále využívají v průmyslu u serverových konzolí pro diagnostiku a také u konfigurace síťových prvků, jako je switch nebo router. Využívají se v těchto oblastech, protože jsou levné a jednoduché.

RS-232 se již nepoužívá u osobních počítačů, protože už je nyní velmi zastaralý, avšak na průmyslových zařízeních se stále používá, včetně jeho potomků RS-422 a RS-485. Od tohoto typu se nejspíš ani upouštět nebude, protože je jednoduchý a stačí k tomu, aby se pouze přenesly nějaké data do a z průmyslového stroje. Řeší se zde pouze tok bitů mezi jedním koncem vodiče a druhým. Konkrétně se hodně využívá u CNC strojů.[30]

TTL logika byla kdysi hodně využívaná, nicméně dnes se od ní opouští z důvodu velké spotřeby. Kdysi se používala například v procesorech počítačů, v tiskárnách nebo v monitorech terminálů. Pomocí logiky TTL se tvořily integrované obvody, ale dnes už se tvoří spíše z logiky CMOS. Nicméně, dnes se ještě používá u některých zařízení z důvodu vysoké přesnosti a menších nákladů. Příkladem dnešního použití jsou právě různé typy smart displejů, CNC strojů, nebo logování průmyslových procesů.[31]

2.4.5 UART

UART je zkratka z angličtiny Universal asynchronous receiver-transmitter. UART je zařízení, které je nejčastěji umístěno přímo na čipu. Zařízení slouží pro uskutečnění komunikace mezi

dvěma asynchronními sériovými zařízeními.

Vysílání a přijímání sériových dat
Vstupem pro UART jsou bajty. Tyto bajty jsou dále převedeny na jednotlivé bity a ty seřadí za sebe do sekvenčního tvaru. Na druhé straně, druhý UART znovu poskládá sérii bitů v jeden bajt. Každý UART má posuvný registr, který je základ pro převod mezi sériovou a paralelní formou. Sériový přenos bitů, přes jednovodičové nebo jiné médium, je levnější, než paralelní přenos přes více vodičů.

Komunikace může být buď simplexová, což znamená, že jedno zařízení pouze vysílá bity a na druhém konci komunikace druhé zařízení ho přijímá. Pro tuto komunikaci je potřeba pouze jeden vodič. Dále může být half-duplexová, kde komunikace probíhá také po jednom vodiči, ale musí se nějakým způsobem domluvit na tom, kdy je jedno zařízení vysílač a druhé přijímač a naopak. A poslední možnost je full-duplex, kde jsou dva vodiče, jeden vodič je vysílač a druhý je přijímač.[32]

2.4.6 RS-232

RS-232 je komunikační standard, který se používá v sériové komunikaci. Umožňuje komunikaci na 15 metrů. Pokud chceme mít komunikační médium delší než 15 metrů, snižujeme tím postupně jeho přenosovou rychlost. Komunikace u RS-232 probíhá asynchronní sériovou komunikací, kde se první přenáší nejméně významný bit (LSB) až po nejvýznamnější bit (MSB). RS-232 podporuje full duplexovou komunikaci a umožňuje přenášet až do 1 Mbps. Zapojit se však může pouze jeden vysílač k jednomu přijímači, neumožňuje tedy propojít více, než dvě zařízení mezi sebou. Jedná se tedy o komunikaci jeden na jednoho (point to point).[33]

2.4.7 TTL

Původně fungovaly TTL okruhy s 5V napěťovým zdrojem, nicméně dnes je běžně používána 3,3V logika, nebo i nižší. Signály se dělí na nízké a vysoké, z angličtiny “low” a “high”. TTL signál, který je definován jako nízky, je u 5V logiky mezi napěťovými úrovněmi 0 V a 0,8 V a vysoký signál je u 5V logiky mezi úrovněmi 2 V a napěťovým zdrojem, což je 5 V. Mezi napěťovými úrovněmi 0,8 V a 2 V je zakázané pásmo. Pokud se napětí na vstupu nachází v tomhle napěťovém pásmu, nepošle se na výstup ani logická 0, ani logická 1. Toto zakázané pásmo funguje jako prevence proti záměně logické 0 a logické 1 a naopak. Kdyby tato logika fungovala tak, že by definovala nízkou úroveň od 0 V do 2,5 V a vysokou úroveň od 2,5 V do 5 V, mohlo by se stát, že jednou by se vstupní signál o úrovni blízké 2,5 V definoval jako nízká úroveň a jindy jako vysoká úroveň, z důvodu vnějšího rušení nebo teploty. U TTL logiky s napěťovým zdrojem 3,3 V toto funguje stejně s nižšími úrovněmi.[34]

2.4.8 I²C

I²C, neboli IIC (Inter-Integrated Circuit), je sběrnice, která se používá k připojení nízkorychlostních periférií k procesorům nebo mikrokontrolérům na krátkou vzdálenost. I²C používá dva obousměrné kanály. Sériovou datovou linku (SDL - Serial Data Line) a sériovou hodinovou linku (SCL - Serial Clock Line). Typické napěťové úrovně jsou 3,3 V a 5 V, ale mohou se použít i jiné úrovně.

I²C komunikuje pomocí referenčního modelu, který je sběrnicový s hodinovým signálem SCL a datovou linkou SDL. V tomto modelu je nutné adresovat zařízení 7-bitovou unikátní adresou. Zařízení připojené na sběrnici může být ve dvou rolích, *master* a *slave*. Mohlo by se zdát, že *master* může být pouze jeden a *slavů* více, nicméně může být i režim *multimaster*, který umožňuje na jedné sběrnici mít více *masterů*. V režimu kde je jeden *master* je implementace jednodušší v tom smyslu, že zde nemůže nastat čekání na ukončení komunikace jiného *mastera*, takže je komunikace i rychlejší.[35]

Master je prvek, který generuje hodinový signál a zahajuje komunikaci se *slavy*, zatímco *slave* je prvek, který pouze přijímá hodinový signál a reaguje na opačný typ zpráv, který nastaví *master*. *Master* i *slave* mají každý 2 módy, jeden pro vysílání a druhý pro přijímání zpráv. *Master* si vždy nastaví, jestli chce přijímat nebo vysílat a *slave* se přizpůsobí opačným typem.[36]

2.4.9 SPI

SPI (Serial Peripheral Interface) je sériové komunikační rozhraní, které se používá na krátké vzdálenosti u vestavěných systémů, SD kart nebo LCD. SPI zařízení komunikuje ve full-duplexu, což je obousměrná komunikace. Používá architekturu *master* a *slave*, kde může být pouze jeden *master* a více *slavů*. V této architektuře je *master* řídicí prvek a *slave* je podřízený uzel.

SPI obsahuje 4 komunikační pracovní vodiče. SCLK (Serial Clock), což je vodič, přes který se vysílá hodinový signál. Hodinový signál vysílá master. MOSI (Master Output Slave Input), je vodič, který slouží k posílání dat směrem od mastera ke slavovi. MISO (Master Input Slave Output), je vodič, který slouží k opačné komunikaci. A poslední vodič, SS (Slave Select), slouží k určení, se kterým podřízeným uzlem bude master komunikovat.[37]

Přenos dat

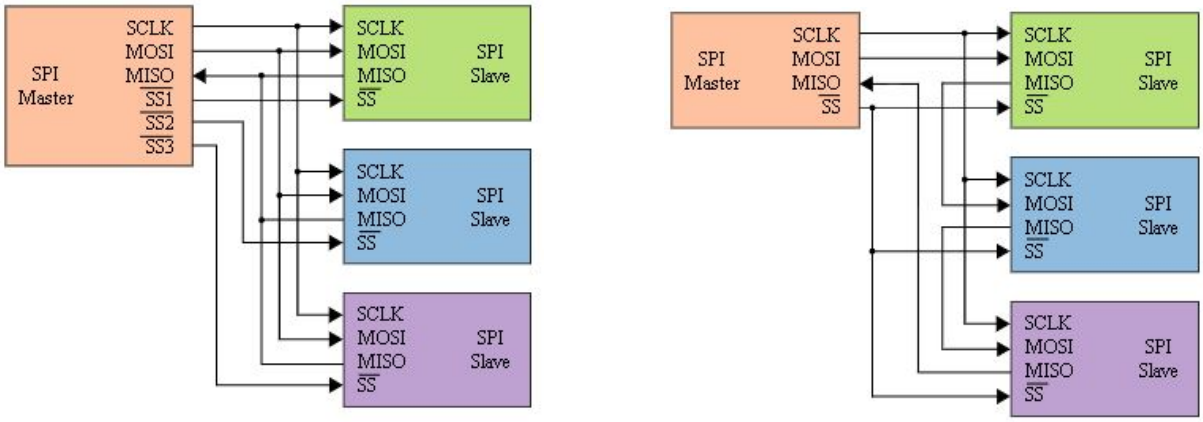
K zahájení komunikace, master nejdříve spustí generování hodinového signálu, na který je připojen i podřízený uzel. Potom master vybere podřízený uzel, který má nastavenou logickou 0 na pinu SS. Během každého hodinového signálu se provede odeslání jednoho bitu. Master vyšle bit po MOSI vodiči, který podřízený uzel přečte a mezitím pošle i podřízený uzel bit po MISO vodiči pro mastera, který přečte.

Slave konfigurace

V konfiguraci typu independent slave je komunikace zajištěna pomocí čipu, který určuje s kte-

rým podřízeným uzlem bude master komunikovat. Tato metoda je u SPI komunikace normálně používána. Master musí obsluhovat takový počet SS výstupů, kolik je připojených podřízených uzlů. U tohoto typu komunikace je zapotřebí, aby byly vodiče třístavové. Stavy jsou klasicky nízká úroveň, vysoká úroveň a třetí stav je vysoká impedance. Nízká a vysoká úroveň se používá normálně a vysoká impedance se používá, když nechceme vybrat daný podřízený uzel. Tento způsob se využívá, protože jsou podřízené uzly propojené. Schéma zapojení independent slave se nachází na obrázku 8a. [38]

U daisy chain konfigurace je pouze jeden SS vodič. Pokud je připojených více podřízených uzlů, tak se informace posílají z mastera na všechny. Data se však posílají v posunuté sekvenci o jeden hodinový signál za každý podřízený uzel. To znamená, že pokud se pošlou data z mastera na podřízené uzly, tak první podřízený uzel získá data v prvním hodinovém signálu, druhý podřízený uzel ve druhém hodinovém signálu, a tak dále. Toto je zapříčiněno z důvodu, že první podřízený uzel má výstup nastavený na vstup druhý podřízený uzel a druhý podřízený uzel na dalšího, a tak dále.[37] Schéma zapojení daisy chain se nachází na obrázku 8b. [39]



(a) Independent Slave

(b) Daisy Chain

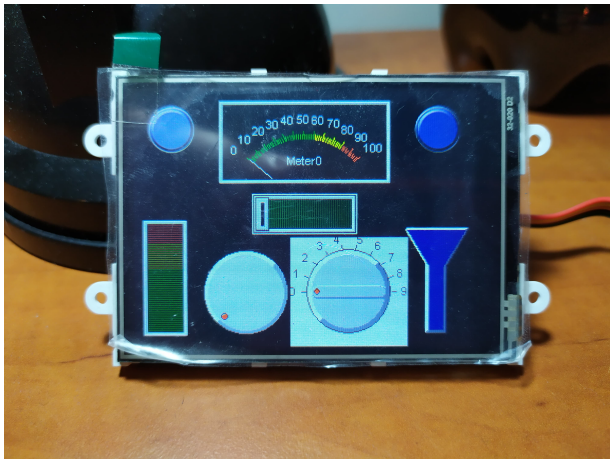
Obrázek 8: Možnosti zapojení SPI

2.5 Parametry vybraných displejů

V této kapitole se nachází podrobnější popis vybraných zástupců smart displejů různých značek. Popsány jsou grafické prvky, způsob komunikace a hardwarová specifikace.

2.5.1 4D Systems uLCD-32PTU

Na obrázku 9 se nachází displej značky 4D Systems[40]. Jedná se o dražší a kvalitnější australské smart displeje. Cena tohoto konkrétního modelu je momentálně 79 dolarů.



Obrázek 9: 3.2 palcový 4D Systems smart displej

HW specifika

Displej má 8 × 16-bitový časovač s přesností až 1 milisekundu. Má také možnost přehrávat zvuky ve formátu WAV, které jsou uložené na micro-SD kartě. Displej obsahuje 4DGL grafiku a knihovnu systémových funkcí, což umožňuje zobrazit v plných barvách obrázky, animace a videa a také podporuje všechny Windows fonty. [41] Detailní tabulka hardwarových specifikací displeje se nachází v příloze, tabulka 15. [41]

Podporované příkazy

Příkazy, které displej podporuje jsou popsány v tabulce 1. [42] Zkratka <oid> značí id objektu, <id> index objektu daného typu, <sum> je kontrolní součet, <val> je hodnota, <len> je délka řetězce a <chr> je znak řetězce.

Tabulka 1: 4D Systems příkazy

Funkce	Příkaz	Délka	Příklad
požadavek o hodnotu	0x00 <oid> <id> <sum>	4 bajty	0x00, 0x04, 0x00, 0x04
zápis hodnoty	0x01 <oid> <id> <val> <val> <sum>	6 bajtů	0x01, 0x04, 0x00, 0x00, 0x10, 0x05
zápis textu	0x02 <id> <len> <chr> <chr> <chr> <sum>	x + 4 bajty	0x02, 0x00, 0x02, 0x61, 0x62, 0x63, 0x05
vrácení hodnoty	0x71 <oid> <id> <val> <val> <sum>	6 bajtů	0x05, 0x04, 0x00, 0x00, 0x10, 0x05
událost	0x07 <oid> <id> <val> <val> <sum>	6 bajtů	0x07, 0x04, 0x00, 0x00, 0x10, 0x05

Zobrazovací prvky

Zobrazovací prvky jsou popsány v příloze v tabulce 11. [42]

Základní informace

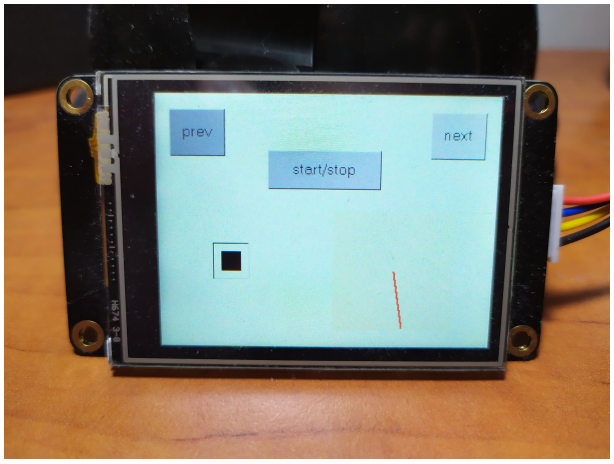
Základní informace o displeji jsou v tabulce 2.

Tabulka 2: 4D Systems uLCD-32PTU základní informace

Vlastnost	Hodnota
Software nutný k provozu	4D Workshop
Komunikační rozhraní	UART
Rozhraní pro nahrání GUI	UART
Přenosová rychlost	9600 baudů
Způsob ovládání	Dotykový displej
Napájení	5 voltů

2.5.2 Nextion enhanced nx3224k024

Na obrázku 10 se nachází smart displej značky Nextion[43]. Typ displeje NX3224K024 je poměrně levný a jednoduchý. Je vhodný pro nenáročné uživatele. Cena tohoto modelu je 20,4 dolarů.



Obrázek 10: 2.4 palcový Nextion smart displej

HW specifika

Detailní hardwarové specifikace se nachází v příloze v tabulce 16. [44]

Podporované příkazy

Jednotlivé příkazy jsou popsány v tabulce 3. Na displej se posílají příkazy ve formě textu zakódovaného do bajtů, přičemž text je podobný syntaxi programovacího jazyka C. Displeje typu

nextion umožňují používat mnoho příkazů, dokonce se na něj mohou posílat celé programy.[45] V tabulce je použita zkratka <id>, což je identifikátor objektu, <val> je hodnota a <text> je text. Za tyto zkratky se doplňují patřičné hodnoty.

Tabulka 3: Nextion příkazy

Funkce	Příkaz	Délka	Příklad
požadavek o hodnotu	get <id>.val	x	get n0.val
zápis hodnoty	<id>.val=<val>	x	n0.val=10
zápis textu	<id>.txt=«text>"	x	t0.txt="abc"
vrácení hodnoty	0x71 <val> <val> <val> <val>	5 bajtů	0x71,0x00,0x00,0x00,0x10
událost	<id> <id> <val> <val> <val> <val>	6 bajtů	0x6e,0x30,0x00,0x00,0x00,0x10

Zobrazovací prvky

Zobrazovací prvky jsou popsány v příloze v tabulce 12. [44]

Základní informace

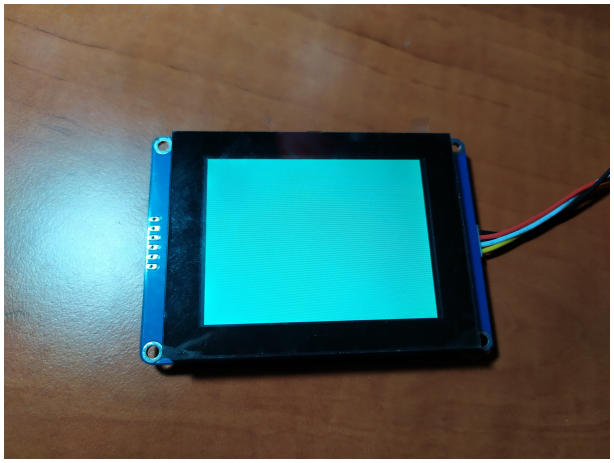
Základní informace o displeji jsou v tabulce 4.

Tabulka 4: Nextion nx3224k024 základní informace

Vlastnost	Hodnota
Software nutný k provozu	Nextion IDE
Komunikační rozhraní	UART
Rozhraní pro nahrání GUI	MicroSD karta
Přenosová rychlost	9600 baudů
Způsob ovládání	Dotykový displej
Napájení	5 voltů

2.5.3 Surenoo h28a-ic

Na obrázku 11 je displej značky Surenoo[46] model h28a-ic. Tato původně čínská firma distribuuje po světě pomocí stránek Aliexpressu pár modelů těchto smart displejů za minimální cenu, která je okolo 10 dolarů.



Obrázek 11: 2.8 palcový Surenoo smart displej

HW specifika

Detailní informace o displeji se nachází v příloze v tabulce 17. [47]

Podporované příkazy

Příkazy jsou posílány v jednotlivých bajtech. Struktura příkazů má různý počet parametrů. Příkazy umožňují většinou pouze číst jednotlivé prvky nebo do nich zapisovat a některé prvky umožňují zaznamenání události. Jednotlivé příkazy jsou popsány v příloze v tabulce 14. [48] Použité zkratky v tabulce jsou <cid>, což je identifikátor prvku, <pid> je identifikátor obrazovky, <text> je text, <val> je hodnota, <sta> je status a <type> je typ události

Zobrazovací prvky

Zobrazovací prvky jsou popsány v příloze v tabulce 13. [49]

Základní informace

Základní informace o displeji jsou v tabulce 5.

Tabulka 5: Surenoo h28a-ic základní informace

Vlastnost	Hodnota
Software nutný k provozu	VisualLcdStudio
Komunikační rozhraní	UART
Rozhraní pro nahrání GUI	I^2C
Přenosová rychlost	9600 baudů
Způsob ovládání	Dotykový displej
Napájení	5 voltů

3 Návrh a implementace software

V této kapitole bude popsán problém se sjednocením komunikace displejů, požadavky na knihovnu a její návrh. Dále budou popsány jednotlivé soubory knihovny a části konfiguračních souborů jednotlivých displejů.

3.1 Problém sjednocení

Problém sjednocení je zapříčiněn tím, že není zavedený žádný komunikační standard, který by sjednocoval komunikaci všech smart displejů. Každá značka smart displejů si řeší komunikaci podle sebe. Například značka 4D Systems řeší komunikaci pomocí hexadecimálních hodnot, kde každý bajt má svou vlastní funkci, označuje určitý typ prvku nebo nese specifickou hodnotu. Naproti tomu značka Nextion používá pro komunikaci zakódovaný text do bajtů. Tento text připomíná programovací jazyk C. Dále je problém, že některý displej může komunikovat připojením UART, jiný podle protokolu I^2C a další podle protokolu SPI. Všechny tyto komunikační protokoly mají jinou strukturu, a tím pádem nastává problém, že pro každý typ komunikace je třeba používat jiný způsob přístupu.

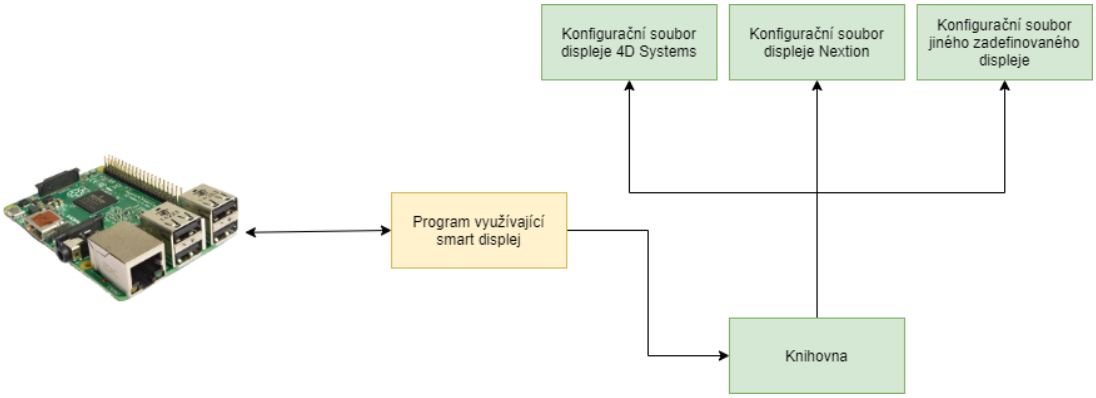
3.2 Požadavky na knihovnu

Jako hlavní požadavek na knihovnu je zjednodušit práci programátorům při tvoření konkrétních programů, které komunikují mezi Raspberry Pi a Smart displeji. Bez této knihovny by v případě, že by došlo k výměně displeje, byla potřeba změnit celá zobrazovací a komunikační část s displejem. Poté, až by pochopili všechny principy, jak displeje komunikují s Raspberry Pi, museli by se naučit příkazy, kterými konkrétní displeje komunikují. Způsob, jak se využívá knihovna je znázorněno na obrázku 12.

Tato knihovna podporuje komunikaci smart displejů 4D Systems a Nextion. Otestována je pouze na jednom displeji každé značky, nicméně konkrétní příkazy jednotlivých značek by měly být totožné napříč různými variantami displejů. Knihovna se soustředí pouze na sériovou komunikaci.

Knihovna byla navrhována tak, aby poskytovala:

- Unifikované rozhraní pro práci s různými chytrými displeji
- Snadnou rozšiřitelnost o nové displeje
- Možnost definice komunikačního rozhraní a parametrů připojeného displeje
- Snadné nastavení podporovaných příkazů displejem.



Obrázek 12: Blokové schéma použití knihovny

3.3 Popis vývoje

Vývoj probíhal v programovacím jazyce Python3, protože má bohatou komunitu, takže existují knihovny, které zajišťují sériové spojení. Dále je v něm jednodušší vývoj než v jiných programovacích jazycích jako je třeba C/C++.

Vývoj probíhal ve vývojovém prostředí Visual Studio Code[50], což je program od společnosti Microsoft, který slouží k editování zdrojových kódů. Díky pluginům, které se mohou k tomuto programu připojit z něj lze udělat dobré vývojové prostředí pro Python3. Další výhodou Visual Studio Code je, že umožňuje editovat zdrojový kód přes SSH, takže je možno psát kód na desktopovém počítači, zatímco kód se bude fyzicky nacházet na Raspberry Pi. Také bylo potřeba zajistit přihlašování k Raspberry Pi pomocí SSH klíče, místo klasického hesla a stáhnout verzi Visual Studio Code pro vzdálený přístup na Raspberry Pi.

3.4 Popis knihovny

Tato knihovna má sloužit k zjednodušení práce s jednotlivými displeji a sjednotit komunikaci napříč různými značkami smart displejů a typy komunikací. Jejím rozhraním jsou metody, jako nastavení hodnoty, požadavek o hodnotu, nebo zjištění události. U knihovny tedy stačí používat tyto metody a není nutné se zabývat složitým tvořením jednotlivých příkazů nebo vytvářením spojení pro daný typ komunikace. Knihovna úzce spolupracuje s výše uvedeným konfiguračním souborem, který definuje způsob, jaký typ komunikace se bude používat, jak se budou tvořit jednotlivé zprávy nebo které prvky daný displej obsahuje. Diagram znázorňující závislosti mezi jednotlivými python soubory je v příloze na obrázku 18. A diagram nasazení je v příloze na obrázku 19.

Třídy v knihovně a jejich vzájemné vazby se nachází v příloze na obrázku 20. Z diagramu tříd je viditelné, že je použita dědičnost, kde bázové třídy jsou IMessageHandler, IMessageDecoder a IConnectionHandler. Tyto abstraktní třídy slouží jako předpis pro odvozené třídy. Je to navrženo tímto způsobem, protože názvy metod v odvozených třídách jsou vždy stejné, pouze se mění jejich implementace. Další důvod je, že když by někdo rozšiřoval tuto knihovnu o další značku

displeje, tak bude přesně vědět, které metody má přepsat z bázové třídy pro správnou funkčnost. Sekvenční diagram metody setValue je v příloze na obrázku 21, sekvenční diagram metody getValue je v příloze na obrázku 22 a sekvenční diagram metody initEventHandler je v příloze na obrázku 23.

Knihovna obsahuje návrhový vzor Factory, který je patrný i z diagramu tříd. Tento návrhový vzor je použitý pro zajištění jednoduché rozšiřitelnosti knihovny. Do třídy DisplayHandler se importuje pouze Factory třída a při volání metody se podle názvu displeje vrátí instance konkrétní odvozené třídy.

Knihovna vytváří závislosti na několika balíčcích. Pro správnou funkčnost knihovny jsou potřeba balíčky json, os.path, serial, smbus, spidev a abc.

json
Balíček json slouží k převedení konfiguračního souboru, který je s příponou json, do proměnné. Tato proměnná bude typu slovník, což je proměnná, která má unikátní klíč, pomocí kterého se přistupuje k hodnotě. Typ slovník je vlastně velmi podobný JSONu, protože hodnoty v JSONu jsou také uložené pomocí klíče a hodnoty.

os.path
Balíček os.path slouží k získání absolutní cesty k aktuálnímu souboru. V knihovně je to použito pro získání cesty ke konfiguračnímu souboru a tato cesta je poté využita k jeho otevření.

serial
Balíček serial se využívá k vytvoření spojení mezi displejem a Raspberry Pi pomocí komunikačního protokolu UART.

smbus
Podobně balíček smbus slouží k zjednodušení spojení mezi displejem a Raspberry Pi pomocí komunikačního protokolu I^2C.

spidev
Balíček spidev slouží k zjednodušenému vytváření spojení mezi displejem a Raspberry Pi pomocí komunikačního protokolu SPI.

abc
Balíček abc slouží pro vytváření abstraktních tříd a abstraktních metod. V knihovně se toto využívá ve třídách IMessageHandler, IMessageDecoder a IConnectionHandler.

3.4.1 Struktura knihovny

Zde je popsána stromová struktura složek knihovny. Z této struktury se dá zjistit, kde je jaký soubor umístěný.

```
library/
├── DisplayHandler.py
├── displayConfigurations/
│   ├── 4dsystems.json
│   └── nextion.json
├── messageHandlers/
│   ├── MessageHandlerFactory.py
│   ├── IMessageHandler.py
│   ├── MessageHandler4dsystems.py
│   ├── MessageHandlerNextion.py
│   └── Checksum.py
├── messageDecoders/
│   ├── MessageDecoderFactory.py
│   ├── IMessageDecoder.py
│   ├── MessageDecoder4dsystems.py
│   └── MessageDecoderNextion.py
└── connectionHandlers/
    ├── ConnectionHandlerFactory.py
    ├── IConnectionHandler.py
    ├── ConnectionHandlerUART.py
    ├── ConnectionHandlerI2C.py
    └── ConnectionHandlerSPI.py
```

3.4.2 DisplayHandler.py

Třída *DisplayHandler* slouží jako rozhraní ke knihovně. Obsahuje důležité metody, jako *setValue*, *getValue* a *initEventHandler*. Metoda *setValue* slouží k nastavení hodnoty, metoda *getValue* k získání číselné hodnoty z prvku a metoda *initEventHandler* se nastaví pro získávání informací o tom, že nastal nějaký uživatelský vstup, neboli událost. Dále také obsahuje méně důležité metody, jako výpis všech prvků, které jsou přítomny na displeji podle konfiguračního soubrou, nebo výpis všech prvků, které může mít displej. Zdrojový kód se nachází ve výpisu 6.

Nejprve se do souboru *DisplayHandler* naimportuje balíček *json*, který bude sloužit k zpracování konfiguračního souboru a balíček *os.path*, který bude sloužit k získání aktuální cesty. Dále se naimportují všechny Factory třídy, které se nachází v podsložkách. Naimportuje se tedy třída *ConnectionHandlerFactory* ze souboru ConnectionHandlerFactory, který je ve složce connecti-

onHandlers. Další třídou je *MessageDecoderFactory*, která je v souboru MessageDecoderFactory, ve složce messageDecoders. Jako poslední třídou je *MessageHandlerFactory*, která je v souboru MessageHandlerFactory, ve složce messageHandlers.

Pro vytvoření instance třídy *DisplayHandler* je potřeba jeden parametr. Tento parametr je název konfiguračního souboru ve složce displayConfigurations. V konstruktoru této třídy se vytvoří proměnná *filePath*, do které se uloží absolutní cesta ke konfiguračnímu souboru přes balíček *os.path* a název souboru předaný v parametru. Dále se otevře soubor pomocí příkazu *with*, který se nachází na cestě proměnné *filePath* v módu čtení. V tomto bloku se do proměnné data uloží obsah souboru pomocí metody *read*. Po opuštění tohoto bloku kódu se do instanční proměnné *displayConf* uloží data z proměnné data pomocí metody *json.loads*. K datům v proměnné *displayConf*, která reprezentuje konfigurační soubor displeje, se nyní může přistupovat jako ke slovníku.

Dále se uloží do instančních proměnných jednotlivé Factory třídy. Do instanční proměnné *msgHandlerFactory* se vytvoří objekt třídy *MessageHandlerFactory*, do instanční proměnné *msgDecoderFactory* se vytvoří objekt třídy *MessageDecoderFactory* a do instanční proměnné *conHandlerFactory* se vytvoří objekt třídy *ConnectionHandlerFactory*. Poté se vytvoří do instančních proměnných konkrétní třídy předchozích Factory tříd. Pomocí instanční proměnné *msgHandlerFactory*, která uchovává objekt *MessageHandlerFactory*, se zavolá metoda *getMessageHandler* s parametrem názvu konkrétního displeje, která vytvoří instanci konkrétní třídy *MessageHandler* a uloží jej do instanční proměnné *msgHandler*. Poté se zavolá metoda *getConnectionHandler* nad instanční proměnnou *conHandler*, která má přiřazený objekt třídy *ConnectionHandlerFactory* s parametry typu komunikace a komunikačního souboru, která slouží pro vytvoření konkrétního typu komunikace a uloží se do proměnné *conHandler*. Nakonec se vytvoří pomocí metody *getMessageDecoder* nad instanční proměnnou *msgDecoder* konkrétní *MessageDecoder* a uloží se do proměnné *msgDecoder*. Tato metoda má dva parametry, název displeje a konfigurační soubor.

getIdsByItem

Metoda *getIdsByItem* slouží k vypsání prvků určitého typu, které se nachází v konfiguračním souboru. Metoda má parametr *item*, což je název typu prvku. Uvnitř metody se vrátí z proměnné *displayConf* hodnota klíče *indexes* z hodnoty klíče určitého prvku předaném v parametru metody z hodnoty klíče *elements*.

getItemsWithId

Metoda *getItemsWithId* slouží k vypsání všech typů prvků, které obsahují alespoň jeden konkrétní prvek v konfiguračním souboru. Uvnitř metody se vytvoří prázdné pole a přiřadí se do proměnné *result*. Poté se vytvoří cyklus, který projde všechny hodnoty klíče *elements* z proměnné *displayConf*. Uvnitř cyklu se zkontroluje, zda klíč *indexes* z hodnot klíče *item* z hodnot klíče *elements* obsahuje nějaké hodnoty. Jestli nějaké hodnoty obsahuje, tak se přidá do pole

result. Až skončí cyklus, tak metoda proměnnou *result* vrátí.

setValue

Metoda *setValue* slouží k poslání zprávy, která změní hodnotu prvku na displeji. Metoda má tři parametry. Parametr *item*, který slouží k definování typu prvku, parametr *id*, který slouží k definování názvu konkrétního prvku a hodnotu, která se má přiřadit do prvku. V těle metody se zkontroluje, o jaký typ hodnoty předané v parametru se jedná. Jestli se jedná o číselnou hodnotu, zavolá se nad proměnnou *msgHandler* metoda *structureSetValMessage* s parametry konfiguračního souboru, názvu typu prvku, názvu konkrétního prvku a hodnoty a odpověď se uloží do proměnné *message*. Pokud se jedná o textovou hodnotu, zavolá se nad proměnnou *msgHandler* metoda *structureTextMessage* s parametry konfiguračního souboru, názvu typu prvku, názvu konkrétního prvku a hodnoty a odpověď se uloží do proměnné *message*. Potom se do proměnné *returnLen* přiřadí délka pole, která se nachází v konfiguračním souboru pod klíčem *returnSetStructure*. Nakonec se přes proměnnou *conHandler* pošle metoda *sendMessage* s parametry *message* a *returnLen*. Sekvenční diagram metody setValue je v příloze na obrázku 21.

getValue

Metoda *getValue* slouží k poslání požadavku o hodnotu na prvku displeje. Metoda má parametry názvu typu prvku a názvu konkrétního prvku. V těle metody se nad proměnnou *msgHandler* zavolá metoda *structureGetValMessage* s parametry konfiguračního souboru displeje, názvu typu prvku a konkrétního názvu prvku, která vrátí zprávu a uloží ji do proměnné *message*. Dále se přiřadí do proměnné *returnLen* délka zprávy, která se nám vrátí, pomocí klíče *returnGetStructure* v proměnné *displayConf* a metody *len*. Potom se zavolá nad proměnnou *conHandler* metoda *sendMessage* s parametry *message* a *returnLen*, která pošle zprávu na displej a přijme odpověď od displeje, kterou uloží do proměnné *response*. Dále se nad proměnnou *mesDecoder* zavolá metoda *decodeToNumber* s parametrem *response* a odpověď se uloží do proměnné *decodedMessage*. Metoda *decodeToNumber* převede zakódovanou zprávu na číslo. Než metoda skončí, vrátí proměnnou *decodedMessage*. Sekvenční diagram metody getValue se nachází v příloze na obrázku 22.

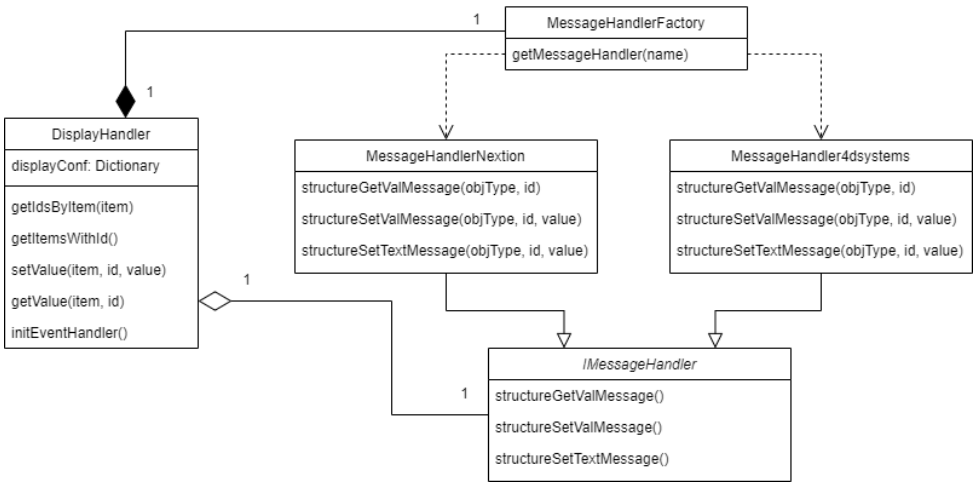
initEventHandler

Metoda *initEventHandler* slouží k zachytávání událostí, které vytvoří uživatel na displeji. V těle metody se zjistí pomocí metody *len* a klíče *returnEventStructure* z konfigračního souboru délka pole a ta se uloží do proměnné *returnLen*. Poté se nad proměnnou *conHandler* zavolá metoda *checkEvent* s parametrem *returnLen*. Metoda *checkEvent* zjistí, zda proběhla nějaká událost a uloži ji do proměnné *response*. Potom se vytvoří proměnná *decodedMessage*, která má v sobě prázdný řetězec. Dále se zkontroluje, jestli proměnná *response* není prázdný řetězec a pokud ne, tak se zavolá nad proměnnou *mesDecoder* metoda *decodeEvent* s parametrem *response*, což vrátí pole, kde se nachází zdroj události a hodnota, na kterou se prvek změnil. Toto pole se uloží do proměnné *decodedMessage* a poté metoda tuto proměnnou vrátí a ukončí se. Sekvenční diagram

této metody se nachází v příloze na obrázku 23.

3.4.3 MessageHandlerFactory.py

Tato třída, *MessageHandlerFactory*, která se nachází ve složce messageHandlers, slouží k vytváření různých typů MessageHandlerů pomocí textu, který se zadá v dané metodě. Třída vytváří instance různých typů MessageHandlerů dle návrhového vzoru factory. Tato třída obsahuje pouze jednu metodu, a tou je *getMessageHandler*. Metoda bere jeden textový parametr, kterým se určí, který typ instance MessageHandleru se vytvoří. Na obrázku 13 se nachází tato část knihovny.



Obrázek 13: Diagram vzoru Factory u třídy MessageHandlerFactory

3.4.4 IMessageHandler.py

V této třídě se musí naimportovat dvě věci z balíčku *abc*. *ABC* a *abstract method*, což slouží k vytvoření abstraktní třídy a abstraktních metod. Třída *IMessageHandler* je tedy abstraktní třída, která má pouze zadefinované metody, které by měly mít zděděné třídy. Mezi tyto zadefinované metody patří *structureGetValMessage*, což je metoda, která slouží k vytvoření požadavku o hodnotu prvku. Další metoda je *structureSetValMessage*, která slouží naopak k vytvoření zprávy, která nastaví číselnou hodnotu do daného prvku. Poslední metodou je *structureSetTextMessage*, která vytváří zprávu nastavení textu do prvku.

3.4.5 MessageHandler4dsystems.py

Třída *MessageHandler4dsystems* slouží pro konkrétní vytvoření zpráv pro 4D Systems značku displejů. U této třídy se nejdříve musí naimportovat abstraktní třída, ze které se bude dědit, tedy *IMessageHandler*. Dále se naimportuje třída *Checksum*, která poskytne výpočet kontrolního součtu pro každý příkaz. Třída tedy dědí z *IMessageHandler* a má prázdný konstruktor.

structureSetValMessage

Tato metoda má čtyři povinné parametry. Prvním parametrem je parametr s názvem *conf*, který se používá pro předání konfiguračního souboru do metody. Druhým parametrem je *objType*, což je název typu objektu, do kterého se bude zapisovat. Třetí parametr s názvem *id* slouží k identifikaci prvku daného typu. A poslední parametr je hodnota, která se má přiřadit do daného prvku a nese název *value*.

Metoda nejdříve vytvoří prázdné pole, do kterého bude postupně zapisovat strukturu příkazu a na konci metody jej vrátí. Poté vytvoří proměnnou typu boolean, která slouží pro zjištění, zda se jedná o první bajt hodnoty nebo druhý bajt hodnoty, protože 4D Systems obsahuje dva bajty hodnot. Dále se provede cyklus, který podle struktury zapsání hodnoty do prvku z konfiguračního souboru zjišťuje, jak seřadit všechny nutné prvky ve zprávě. Cyklus tedy prochází strukturu a podle názvu prvků připisuje bajty v podobě čísla do předem vytvořeného pole. Všechny potřebné parametry byly již předané při volání metody. Je možné si tedy zjistit podle názvu objektu jeho kód z konfiguračního souboru. Na konci cyklu se vypočte kontrolní součet celé zprávy a přidá se do pole bajtů. Po dokončení cyklu se pole bajtů vrátí.

structureSetTextMessage

Metoda *structureSetTextMessage* funguje velmi podobně jako metoda *structureSetValMessage*. Taktéž bere jako parametry konfigurační soubor, název typu objektu, index objektu daného typu a hodnotu, nicméně tentokrát je hodnota řetězec znaků.

Metoda nejprve vytvoří proměnnou, která bude prázdné pole a do ní postupně bude přidávat čísla. Dále se v cyklu projde struktura z konfiguračního souboru pro vytvoření textové zprávy. Postupně se pro každou položku přidá číselná hodnota do předem vytvořeného pole zprávy. Jakmile cyklus narazí na délku zprávy, změří se délka textu a vloží se výsledný počet znaků do pole. Pokud se v iteraci má zrovna přiřadit text do číselného pole, vytvoří se další cyklus, který projde celý text po znaku a postupně vkládá číselné hodnoty charakteru do výsledného pole. Nakonec se vypočte kontrolní součet pomocí třídy *Checksum* a přiřadí se do pole. Metoda poté vrátí výslednou zprávu v číselných hodnotách pole.

structureGetValMessage

Metoda *structureGetValMessage* slouží k vytvoření zprávy pro požadavek o hodnotu prvku. Metoda potřebuje předat konfigurační soubor pro zjištění struktury dané zprávy, název objektu, aby metoda mohla z konfiguračního souboru zjistit kód pro daný objekt a nakonec *id* pro identifikaci prvku.

Metoda opět nejdříve vytvoří prázdné pole. Poté se projde struktura pro vytvoření požadavku o hodnotu z konfiguračního souboru a v každé iteraci se přidávají do pole odpovídající číselné hodnoty. Při iteraci, kde se požaduje typ objektu, se vybere kód objektu z konfiguračního souboru podle jeho názvu předaném v parametru metody. Pro statické příkazy, jako je kód pro požadavek o hodnotu prvku, se projdou příkazy v konfiguračním souboru. Poté se přiřadí kontrolní součet dané zprávě a přidá se do výsledného pole. Nakonec metoda pole vrátí.

3.4.6 MessageHandlerNextion.py

Třída *MessageHandlerNextion* slouží jako konkrétní třída pro vytváření zpráv pro značku displejů Nextion. Nejdříve se naimportuje abstraktní třída ze souboru IMessageHandler, třída *IMessageHandler*. Z naimportované třídy bude potom dědit třída *MessageHandlerNextion*. Pro vytvoření instance třídy *MessageHandlerNextion* není potřeba žádný předaný parametr, má tedy bezparametrický konstruktor.

structureSetValMessage

Metoda *structureSetValMessage* potřebuje k funkčnosti čtyři parametry. Mezi tyto parametry patří konfigurační soubor, typ objektu, id objektu a hodnota. Konfigurační soubor potřebuje pro zjištění struktury pro zápis hodnoty do prvku a pro přístup k některým statickým příkazům, které nejsou předány v parametrech metody. Typ objektu potřebuje pro zjištění, o jaký objekt se jedná, id objektu potřebuje pro identifikaci objektu, do kterého má zapisovat hodnotu, která je předána jako poslední parametr.

Metoda nejdříve vytvoří řetězec bajtů, který bude sloužit pro udržení struktury zprávy. Dále se prochází v cyklu struktura zprávy pro zapsání hodnoty do prvku z konfiguračního souboru. Při každé iteraci se do řetězce bajtů zakóduje určitý řetězec znaků. Při iteraci, kde se požaduje zápis id objektu, nebo hodnoty, se zakóduje parametr metody a připíše se do výsledné zprávy. Při ostatních iteracích se prozkoumává, zda není příkaz v konfiguračním souboru. Jakmile se celá zpráva zformuje, musí se připojit na konec zprávy tři bajty s hodnotou 255. Vytvoří se tedy proměnná, do které se uloží bajt s hodnotou 255 a poté se spojí se zprávou. Po ukončení cyklu metoda vrátí zprávu.

structureSetTextMessage

Metoda *structureSetTextMessage* slouží k nastavení textu do textového prvku. Metoda potřebuje jako parametry konfigurační soubor, typ objektu, identifikátor objektu a hodnotu.

Metoda neprve vytvoří prázdný řetězec. Poté vytvoří cyklus, který prochází strukturu nastavení textu z konfiguračního souboru. V tomto cyklu jsou všechny části zprávy zakódovány do řetězce a připsány na konec dosavadní celkové zprávy. Většina statických příkazů se nachází v konfiguračním souboru, takže se hledají tam. Pokud je potřeba zadat *id*, tak se vezme parametr metody a připíše se do zprávy. Pokud je třeba zakódovat požadovaný text, tak se vezme z parametru metody. U této části se však nesmí zapomenout, že text se píše do uvozovek, například *t0.txt="pozadovany text"*. Opět se přidá konec zprávy třemi bajty s hodnotou 255. Návratovou hodnotu zprávy tvoří nová zakódovaná zpráva pro displej.

structureGetValMessage

Metoda *structureGetValMessage* má za úkol udělat strukturu zprávy, která požádá o číselnou hodnotu prvku. Metoda potřebuje jako parametry opět konfigurační soubor, typ objektu a jeho

identifikátor.

Metoda opět nejdříve vytvoří prázdný řetězec. Dále se vytváří cyklus, který prochází strukturu pro požadavek o hodnotu prvku z konfiguračního souboru, ve kterém se postupně připisují do řetězce zakódované bajty. Pokud struktura požaduje identifikátor, převezme si jej z parametru, jinak jsou ostatní příkazy statické, čili se nachází v konfiguračním souboru. Na konec zprávy se zase musí připsat tři bajty o hodnotě 255. Nakonec metoda vrátí zprávu zformulovanou jako požadavek o číselnou hodnotu prvku.

3.4.7 Checksum.py

Třída *Checksum* slouží pro vypočtení kontrolního součtu u zprávy displejů, značky 4D Systems. Třída obsahuje pouze jednu metodu, *calculateChecksum*. Tato metoda potřebuje k funkčnosti jeden parametr, a tím je pole číselných hodnot. Metoda vrací vypočtený kontrolní součet tak, že na všechny prvky pole postupně aplikuje logickou funkci XOR a vrátí výsledek. Diagram aktivit této metody se nachází v příloze na obrázku 24.

Uvnitř metody se nastaví proměnná na první prvek předaného pole, která bude reprezentovat výsledek kontrolního součtu. Dále se provede cyklus, který postupně aplikuje logickou funkci XOR a uloží výsledek do předem vytvořené proměnné. Až se projdou všechny prvky, cyklus skončí. Na konci metody se vrátí kontrolní součet všech prvku, který se do metody poslal parametrem. Příklady vstupů a výstupů jsou ve výpise 1.

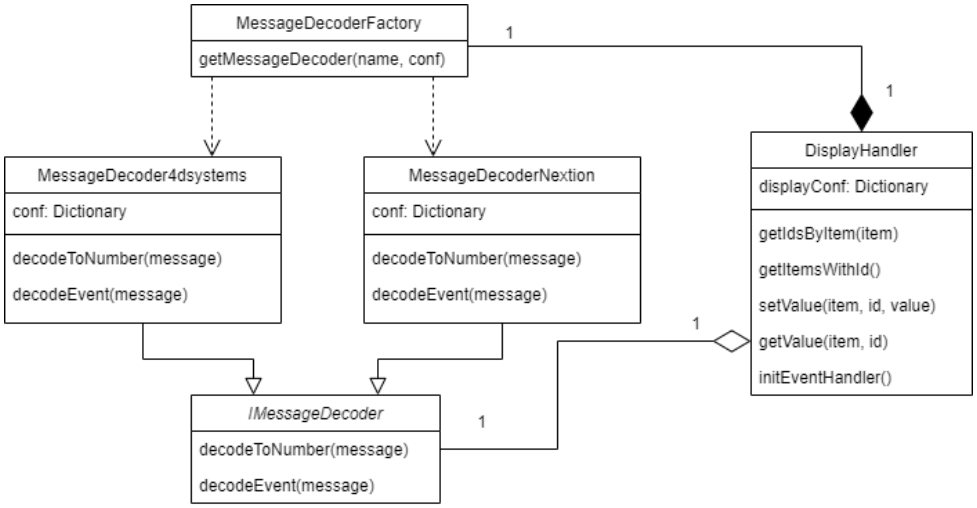
```
Checksum.calculateChecksum([0x01,0x02,0x03,0x04])
#4

Checksum.calculateChecksum([0x02,0x00,0x02,0x61])
#97

Checksum.calculateChecksum([2, 0, 10, 97, 104, 111, 106, 32, 115, 118, 101,
    116, 101])
#85
```

Výpis 1: Příklady vstupů a výstupů metody calculateChecksum

3.4.8 MessageDecoderFactory.py

U třídy *MessageDecoderFactory* se musí nejdříve naimportovat všechny konkrétní *MessageDecoder* třídy. Naimportuje se tedy ze souboru MessageDecoderNextion třída *MessageDecoderNextion* a ze souboru MessageDecoder4dsystems třída *MessageDecoder4dsystems*. Poté se vytvoří třída s názvem *MessageDecoderFactory*, ve které bude jedna metoda s názvem *getMessageDecoder*.

Metoda *getMessageDecoder* má dva parametry. První parametr je název konktétního typu *MessageDecoderu* a druhý parametr je konfigurační soubor. Uvnitř metody se kontroluje, o jaký název konkrétního typu *MessageDecoderu* se jedná a podle toho se vytvoří a vrátí instance konkrétního typu *MessageDecoderu*. Pokud se název neshoduje s žádným konkrétním typem, vrátí se prázdný objekt. Část knihovny se vzorem Factory je na obrázku 14.



Obrázek 14: Diagram vzoru Factory u třídy MessageDecoderFactory

3.4.9 IMessageDecoder.py

V tomto souboru se nejprve naimportuje z balíčku *abc ABC* a *abstractmethod*, což slouží k tomu, aby se mohla vytvořit abstraktní třída a abstraktní metoda. Poté se vytvoří třída *IMessageDecoder*, která dědí ze třídy *ABC*. Uvnitř této třídy jsou dvě metody, *decodeToNumber* a *decodeEvent*, které mají nadepsanou anotaci *abstractmethod*, což značí abstraktní metodu. Obě tyto metody požadují parametr *message*, což je zpráva, která se zde musí předat pro to, aby byla zpráva dekódována. Metoda *decodeToNumber* v této třídě slouží jako předpis pro metodu, která bude převádět zprávu na konkrétní hodnotu a metoda *decodeEvent* v této třídě slouží jako předpis pro metodu, která bude převádět událost na prvek a hodnotu.

3.4.10 MessageDecoder4dsystems.py

Třída *MessageDecoder4dsystems* slouží jako konkrétní třída *MessageDecoderu*. Ve třídě se nejdřív musí naimportovat ze souboru IMessageDecoder třída *IMessageDecoder*, ze které bude třída *MessageDecoder4dsystems* dědit. Vytvoří se tedy tato třída, která dědí z naimportované třídy. V konstruktoru se nastavuje do proměnné *conf* konfigurační soubor, který se předává při vytváření instance této třídy.

decodeToNumber

Metoda *decodeToNumber* slouží k dekódování zprávy a převedení na číselný typ. Metoda požaduje pouze jeden parametr a tím je zpráva. Diagram aktivit je v příloze na obrázku 25.

Metoda nejdříve převede zprávu předanou v parametru na pole jednotlivých bajtů reprezentovaných hexadecimálním číslem. Jednotlivé položky pole jsou ve formátu řetězce a uloženy do proměnné *messageArray*. Poté se vytvoří prázdné pole s názvem *valueArray*, které bude sloužit pro pozdější naplnění hodnotovými bajty. Dále se přiřadí do proměnné *structureArray* struktura zprávy vrácení hodnoty po požadavku z konfiguračního souboru, který je instanční proměnnou. Pak se vytvoří proměnná pro přiřazení výsledku a nastaví se na prázdnou hodnotu.

Potom se vytvoří cyklus, který jde od nuly po počet prvků v poli *structureArray*. V tomto cyklu se kontroluje, zda se aktuální položka nerovná položce hodnoty v struktuře zprávy. Pokud ano, tak se vloží hodnota z pole *messageArray* do pole *valueArray*. Po skončení cyklu se kontroluje zda je v konfiguračním souboru nastavena uspořádání bitů prvku na LSB nebo MSB. Pokud je uspořádání bitů nastaveno na LSB, tak se pole *valueArray* projde pozpátku a spojí se jednotlivé prvky v jeden řetězec a poté se převede na číslo. Pokud je uspořádání bitů nastaveno na MSB, tak se pole prochází od začátku do konce a spojí se jednotlivé prvky v jeden řetězec a poté se převedou na číslo. U obou případů se číselná hodnota uloží do proměnné *result* a poté proměnnou *result* metoda vrátí. Příklady vstupů a výstupů jsou ve výpise 2.

```
msgDecoder.decodeToNumber("050400001011")
#16

msgDecoder.decodeToNumber("050401000001")
#0

msgDecoder.decodeToNumber("050702006465")
#100
```

Výpis 2: Příklady použití decodeToNumber ve třídě MessageDecoder4dsystems

decodeEvent

Metoda slouží k dekódování zprávy, která nastala při nějaké události, například při kliknutí na tlačítko. Výstupem metody je pole, ve kterém je typ prvku, index prvku a jeho hodnota. Metoda požaduje jeden parametr a tím je zpráva. Diagram aktivit se nachází v příloze na obrázku 26.

Metoda nejdříve vytvoří proměnnou *messageArray*, což je pole, které je vytvořené ze zprávy předané v parametru metody. Toto pole má formát řetězce a je rozdělené na jednotlivé bajty. Dále se vytvoří prázdné pole s názvem *valueArray*, které bude později sloužit pro naplnění hodnotovými bajty. Potom se vytvoří pole struktury s názvem *structureArray*, ve kterém bude struktura zprávy o události z konfiguračního souboru. Pak se vytvoří dvě proměnné s názvem *objId* a *objType*. Proměnná *objId* bude sloužit jako hodnota indexu prvku a *objType* bude sloužit jako identifikátor typu objektu. Oběma proměnným se nastaví hodnota prázdného řetězce. Další proměnnou, která se vytvoří je proměnná *result*, která se nastaví na prázdnou hodnotu.

Po zavedení všech proměnných se vytvoří cyklus, který jde od nuly po délku pole *structureArray*. V tomto cyklu se kontroluje zda se aktuální index rovná ve struktuře jednomu ze tří prvků. Tyto tři prvky jsou *value*, *objectType* a *objectId*. Pokud se rovná hodnotě *value*, tak se přidá daná hodnota z pole *messageArray* do pole *valueArray*. Pokud se rovná hodnotě *objectType*, tak se z pole *messageArray* přiřadí do proměnné *objType* a pokud se rovná hodnotě *objectId*, tak se přiřadí do proměnné *objId* z pole *messageArray*. Po skončení cyklu se kontroluje z konfiguračního souboru uspořádání bitů. Pokud je uspořádání bitů LSB, tak se prochází pole *valueArray* pozpátku a vytvoří se z něj řetězec, který se poté převede na číslo a uloží do proměnné *result*. Pokud je uspořádání bitů MSB tak se pole prochází normálně, spojuje se v jeden řetězec a poté se převede na číslo, které se uloží také do proměnné *result*. Nakonec se vytvoří pole *resultArr*, ve kterém jsou hodnoty tří proměnných, *objType*, *objId* a *result*. Metoda toto pole vrátí jako výsledek. Příklady vstupů a výstupů jsou ve výpise 3.

```
msgDecoder.decodeEvent("070400001011")
#["04","00",16]

msgDecoder.decodeEvent("070401000001")
#["04","01",0]

msgDecoder.decodeEvent("070702006465")
#["07","02",100]
```

Výpis 3: Příklady použití decodeEvent ve třídě MessageDecoder4dsystems

3.4.11 MessageDecoderNextion.py

V souboru MessageDecoderNextion se nachází třída *MessageDecoderNextion*, která slouží jako konkrétní třída ze skupiny *MessageDecoder*, pro dekódování zpráv. Ve třídě se nejdříve naimportuje třída, ze které bude třída *MessageDecoderNextion* dědit. Naimportuje se tedy ze souboru IMessageDecoder třída *IMessageDecoder*. Dále se vytvoří třída *MessageDecoderNextion*, která dědí ze třídy *IMessageDecoder*. Pro vytvoření instance této třídy je třeba předat jako parametr konfigurační soubor, který se poté v konstruktoru přiřadí do proměnné *conf*.

decodeToNumber

Metoda *decodeToNumber* slouží k dekódování zprávy a převedení na číselnou hodnotu. Metoda má pouze jeden parametr a tím je zpráva. Diagram aktivit metody decodeToNumber se nachází v příloze na obrázku 27.

Na začátku metody se vytvoří pole *messageArray*, ve kterém se rozdělí zpráva předaná z parametru na jednotlivé bajty. Tyto bajty jsou poté uchovány v řetězcovém formátu. Dále se vytvoří prázdné pole s názvem *valueArray*, do kterého se budou zapisovat hodnotové bajty. Další vytvořené pole je *structureArray*, do kterého se uloží struktura zprávy vrácení hodnoty

po požadavku o ni z konfiguračního souboru. Dále se vytvoří proměnná bez hodnoty s názvem *result*, do které se potom bude zapisovat konkrétní hodnota.

Za další se vytvoří cyklus, který jde od nuly po délku proměnné *structureArray*. V tomto cyklu se kontroluje zda aktuální číslo indexu v poli *structureArray* má hodnotu *value*. Pokud ano, připíše se do pole *valueArray* hodnota z pole *messageArray* s daným indexem. Až skončí cyklus, tak se kontroluje uspořádání bitů v konfiguračním souboru. Pokud je uspořádání bitů LSB, tak se prochází pole *valueArray* pozpátku, spojí se v jeden řetězec a potom se převede na číslo, které se uloží do předem vytvořené proměnné *result*. Pokud je uspořádání bitů MSB, tak se prochází pole normálně, spojí se v jeden řetězec a potom se převede na číslo, které se uloží do předem vytvořené proměnné *result*. Nakonec se proměnná *result* vrací jako výsledek metody. Příklady vstupů a výstupů jsou ve výpise 4.

```
msgDecoder.decodeToNumber("7100000010ffffff")
#16

msgDecoder.decodeToNumber("7100000000ffffff")
#0

msgDecoder.decodeToNumber("7100000064ffffff")
#100
```

Výpis 4: Příklady použití decodeToNumber ve třídě MessageDecoderNextion

decodeEvent

Metoda *decodeEvent* slouží pro dekódování události a vrácení prvku a hodnoty. Metoda požaduje jeden parametr s názvem *message*, což je zpráva, kterou chceme dekódovat. Diagram aktivit metody decodeEvent se nachází v příloze na obrázku 28.

V těle této metody se neprve vytvoří proměnná *messageArray*, což je pole, které má rozdělenou předanou zprávu v parametru na jednotlivé bajty ve formátu řetězce. Dále se vytvoří další pole s názvem *valueArray*. Poté se vytvoří pole *structureArray*, kterému se přiřadí struktura zprávy o události z konfiguračního souboru. Potom se vytvoří proměnná *objId*, která se nastaví na prázdný řetězec. Tak samo jako předchozí proměnná se nastaví i proměnná *objType*. Nakonec se vytvoří proměnná s názvem *result*, která je bez hodnoty.

Dále se v metodě zkontroluje, zda je délka pole *messageArray* stejná jako délka pole *structureArray*. Toto se kontroluje, protože při kliknutí na tlačítko se nevrací hodnota, ale jen konkrétní prvek, na který se kliklo.

Pokud se tedy délky rovnají, vytvoří se cyklus, který jde od nuly do délky pole *structureArray*. V těle cyklu se kontroluje zda hodnota v poli na aktuálním indexu je buď *value*, *objectType*, nebo *objectId*. Pokud se hodnota rovná *value*, tak se přidá do pole *valueArray* hodnota z *messageArray* na aktuálním indexu. Pokud se hodnota rovná *objectType*, tak se hodnota z aktuálního indexu cyklu převede na číselnou hodnotu a ta se poté převede na hodnotu charakteru a uloží do

proměnné *objType*. To samé se provede, pokud se hodnota v aktuálním indexu cyklu rovná *objectId*, s tím rozdílem, že se hodnota uloží do proměnné *objId*. Až se projde celá struktura, tak cyklus skončí. Dále se kontroluje uspořádání bitů. Pokud je uspořádání bitů v konfiguračním souboru nastavena na LSB, tak se prochází pole *valueArray* pozpátku, jeho prvky se složí v jeden řetězec a poté se převede na číselnou hodnotu, která se uloží do proměnné *result*. Pokud je uspořádání bitů v konfiguračním souboru nastavena na MSB, tak se prochází pole *valueArray* od prvního prvku po poslední, složí se v jeden řetězec a poté se převede na číselnou hodnotu, která se uloží do proměnné *result*.

Pokud se délky polí nerovnají, tak se vezme první prvek pole *messageArray*, který se převede na číselnou hodnotu a poté na charakter a uloží se do proměnné *objType*. Dále se vezme druhý prvek pole *messageArray*, převede se na číselnou hodnotu a poté na charakter a uloží se do proměnné *objId*.

Nakonec se vytvoří pole, ve kterém budou dva prvky. První prvek je složení prvků *objType* a *objId* a druhý prvek je *result*. pokud proměnná *result* nebyla změněna od její inicializace, tak zůstává hodnota None. Metoda před ukončením vrací toto pole. Příklady vstupů a výstupů jsou ve výpise 5.

```
msgDecoder.decodeEvent("6e3000000010ffffff")
#["n0",16]

msgDecoder.decodeEvent("6231ffffff")
#["b1",None]

msgDecoder.decodeEvent("7a3300000064ffffff")
#["z3",100]
```

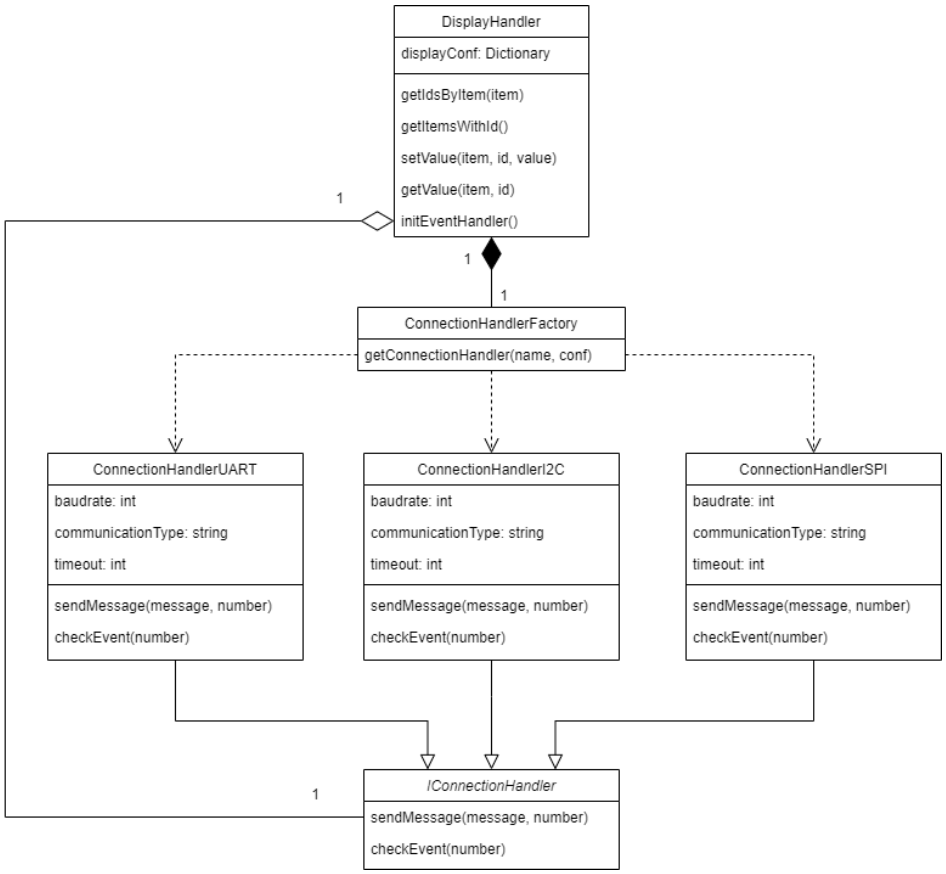
Výpis 5: Příklady použití decodeEvent ve třídě MessageDecoderNextion

3.4.12 ConnectionHandlerFactory.py

V souboru ConnectionHandlerFactory se nachází třída *ConnectionHandlerFactory*, ve které se vytváří konkrétní instance daných typů připojení, tedy *ConnectionHandlerů*. Nejprve se naimportují konkrétní typy připojení. Ze souboru ConnectionHandlerUART se naimpotuje třída *ConnectionHandlerUART*, ze souboru ConnectionHandlerI2C se naimportuje třída *ConnectionHandlerI2C* a ze souboru ConnectionHandlerSPI se naimportuje třída *ConnectionHandlerSPI*. Dále se vytvoří třída *ConnectionHandlerFactory*.

Ve třídě *ConnectionHandlerFactory* je pouze jedna metoda, *getConnectionHandler*. Tato metoda má dva parametry. První parametr je název typu připojení a druhý parametr je konfigurační soubor. Uvnitř metody se zkoumá, o jaký název typu připojení se jedná. Jestli se tedy název rovná řetězci UART, vrátí se instance třídy *ConnectionHandlerUART*. Pokud se rovná

I2C, vrátí se instance třídy *ConnectionHandlerI2C* a jestli se rovná název SPI, tak se vrátí instance třídy *ConnectionHandlerSPI*. Do všech těchto tříd se předává jako parametr při vytváření konfigurační soubor. Pokud se název nerovná žádnému z uvedených řetězců, vrátí se prázdný objekt. Část knihovny se vzorem Factory je na obrázku 15.



Obrázek 15: Diagram vzoru Factory u třídy ConnectionHandlerFactory

3.4.13 IConnectionHandler.py

Třída *IConnectionHandler* v souboru IConnectionHandler slouží jako abstraktní třída pro konkrétní instance tříd ConnectionHandlerů. Nejprve se tedy naimportuje z balíčku *abc ABC* a *abstractmethod*. Poté se vytvoří třída *IConnectionHandler*, která dědí z naimportované třídy *ABC*.

Uvnitř této třídy se nachází dvě metody, *sendMessage* a *checkEvent*. Obě tyto metody mají nadepsanou anotaci *abstractmethod*, což znamená, že se jedná o abstraktní metodu. Těla obou těchto metod jsou prázdná, takže se jedná pouze o předpis. Metoda *sendMessage* navíc požaduje parametr *message*, což je zpráva, která se bude v konkrétních instancích tříd posílat.

3.4.14 ConnectionHandlerUART.py

V souboru ConnectionHandlerUART se nachází třída *ConnectionHandlerUART*. Tato třída slouží k navázání spojení a poslání již zakódované zprávy a přijetí odpovědi. Nejprve se tedy naimportuje balíček *serial*, který slouží k jednoduššímu vytvoření spojení pomocí UARTu. Dále se naimportuje ze souboru IConnectionHandler abstraktní třída *IConnectionHandler*, která slouží jako předpis pro tuto třídu. Dále se vytvoří třída *ConnectionHandlerUART*, která dědí z naimportované třídy *IConnectionHandler*.

Při vytváření této třídy se předává jeden parametr, konfigurační soubor. V konstruktoru této třídy se pak nastavují tři proměnné, *baudrate*, *communicationInterface* a *timeout*. *baudrate* slouží k zjištění, jakou přenosovou rychlostí zařízení komunikuje. *communicationInterface* slouží k zjištění, na jakém rozhraní se má poslouchat či vysílat. A nakonec *timeout* je doba nečinnosti linky, po které se ukončí komunikace.

sendMessage

Metoda *sendMessage* slouží k poslání zprávy konkrétním typem komunikace na konkrétní rozhraní. Po odeslání zprávy metoda přijme odpověď a vrátí ji. Metoda má dva parametry. Prvním parametrem je *message*, což je zpráva a druhý parametr je počet bajtů s názvem *numberOfBytes*, které očekává jako odpověď. Tento parametr je však nepovinný a pokud není zadán, je nastaven na hodnotu None.

Na začátku metody se vytvoří proměnná *response*, která v sobě má prázdný řetězec. Tato proměnná bude sloužit k vrácení odpovědi po zadání zprávy. Poté se vytvoří sériové spojení přes příkaz *with* a metodu *Serial* z balíčku *serial* pod názvem *ser*. Do metody se zadají parametry komunikačního rozhraní, přenosové rychlosti, čas ukončení nečinné linky, parita, stopbity a velikost bajtu. Uvnitř tohoto bloku kódu se pošle zpráva přes metodu *write* a do parametru se předá zpráva, která se předala v parametru metody. Dále se zkontroluje, zda byl zadán druhý parametr metody. Jestli byl parametr zadán, tak se zavolá metoda *read* nad proměnnou *ser* a do parametru se předá druhý parametr metody, *numberOfBytes*, a převede se pomocí metody *hex* na řetězec a zapíše se do proměnné *response*. Po přečtení zadaného počtu bajtů se komunikace automaticky ukončí. Pokud parametr nebyl zadán, tak se zavolá metoda *readline* nad proměnnou *ser*, převede se pomocí metody *hex* na řetězec a zapíše se do proměnné *response*. Při metodě *readline* se spojení neukončí do té doby, než se nepřečtou všechny bajty, a linka bude nečinná alespoň na dobu, která se nastavila. Po ukončení komunikace metoda vrátí proměnnou *response*.

checkEvent

Metoda *checkEvent* slouží pro zachycení události, která může nastat při komunikaci. Metoda má jeden parametr s názvem *numberOfBytes*, neboli počet bajtů, který slouží k zastavení komunikace po obržení tohoto počtu bajtů. Tento parametr je nepovinný a jestli se nezadá, tak se nastaví na hodnotu None.

Uvnitř těla metody se vytvoří proměnná *response*, do které se zapíše zachycená událost. Dále se pomocí příkazu *with* a metody *Serial* z balíčku *serial* vytvoří sériové spojení. Do metody *Serial* se dají parametry komunikačního rozhraní, přenosové rychlosti, čas ukončení nečinné linky, parita, stopbity a velikost bajtu. V tomto bloku kódu se zkontroluje, zda byl zadán parametr metody. Pokud byl zadán, tak přečte daný počet bajtů pomocí metody *read*, převede pomocí metody *hex* na řetězec a uloží do proměnné *response*. Pokud nebyl zadán parametr *numberOfBytes*, tak se pomocí metody *readline* nad proměnnou *ser* čte do té doby, než se neukončí spojení. Výsledná zpráva se převede na řetězec pomocí metody *hex* a zapíše do proměnné *response*. Po ukončení spojení vrátí metoda proměnnou *response*.

3.4.15 ConnectionHandlerI2C.py

Třída *ConnectionHandlerI2C* slouží k vytvoření spojení na základě komunikačního protokolu I2C. V tomto souboru se nejprve naimportuje balíček *smbus*, který usnadňuje práci s I2C protokolem. Dále se naimportuje ze souboru IConnectionHandler abstraktní třída *IConnectionHandler*. Dále se vytvoří třída *ConnectionHandlerI2C*, která dědí z naimportované abstraktní třídy *IConnectionHandler*.

Třída *ConnectionHandlerI2C* má pouze nadepsané metody, které nemají tělo. Toto slouží pro budoucí rozšiřování.

3.4.16 ConnectionHandlerSPI.py

Třída *ConnectionHandlerSPI* slouží jako konkrétní třída pro vytvoření spojení pomocí komunikačního protokolu SPI. V souboru se nejprve naimportuje balíček *spidev*, který slouží pro práci s komunikačním protokolem SPI v Pythonu. Dále se naimportuje ze souboru IConnectionHandler abstraktní třída *IConnectionHandler*, ze které bude třída *ConnectionHandlerSPI* dědit. Vytvoří se tedy třída *ConnectionHandlerSPI* a nastaví se ji dědičnost ze třídy, která se naimportovala.

Ve třídě *ConnectionHandlerSPI* se nachází konstruktor a dvě metody, *sendMessage* a *checkEvent*. Konstruktor a metody mají však prázdná těla, která slouží pouze jako předpis pro budoucí rozšíření.

3.5 Struktura konfiguračního souboru

Konfigurační soubor slouží k sjednocení rozdílů různých značek smart displejů. Tyto rozdíly se musí sjednotit, aby se mohlo jednodušeji pracovat s různými typy smart displejů. Konfigurační soubor obsahuje 4 hlavní části. V první části jsou základní informace o displeji, jako je název nebo přenosová rychlost. Druhá část obsahuje příkazy, které se používají pro komunikaci s displejem. Třetí část má strukturu všech možných typů zpráv, které můžeme buď poslat na displej nebo přijmout od displeje. A poslední, čtvrtá část obsahuje jednotlivé prvky, které se mohou na displeji vytvořit. Obsah konfiguračního souboru displeje 4D Systems je ve výpisu 7 a obsah konfiguračního souboru Nextion ve výpisu 8.

3.5.1 První část

V první části se nachází důležité informace pro vytvoření správného připojení. První řádek slouží k identifikování názvu displeje a tedy i souboru. Název displeje slouží v knihovně k sjednocení komunikačních zpráv. Druhý řádek obsahuje typ komunikace. Toto se dále bude používat pro rozhodnutí, jaký typ připojení se v knihovně použije. Řádek s baudrate obsahuje přenosovou rychlost, která musí být sjednocená pro správnou komunikaci. Communication Interface je řádek, který slouží pro definování, na jaké Raspberry Pi rozhraní bude komunikace chodit. Poslední řádek je uspořádání bitů. Většina displejů posílá více bajtů s hodnotami a tak se musí vědět jestli je umisťuje od nejméně významného po nejvíce významný, nebo od nejvíce významného po nejméně významný. Displeje posílají více hodnotových bajtů, kvůli prvkům, které mohou mít více než 256 hodnot, což je maximální počet stavů v jednom bajtu.

3.5.2 Druhá část

V druhé části jsou příkazy, které se často používají pro komunikaci mezi displejem a Raspberry Pi. Příkaz read slouží pro čtení, write pro zápis hodnoty, writeStr pro zápis textové hodnoty, writeStru pro zápis textové hodnoty, kde jeden znak reprezentují 2 bajty, return je navrácená hodnota po požadavku o ní, ACK je potvrzení o správném přečtení příkazu, event je informace o události a NACK je nesprávné přečtení příkazu.

3.5.3 Třetí část

Třetí část obsahuje jednotlivé struktury sérií příkazů, které mohou v komunikaci mezi displeji a Raspberry Pi nastat. Část příkazů se pojí k druhé části konfiguračního souboru. U displeje značky 4D Systems reprezentuje každá hodnota v poli jeden bajt. U displejů značky Nextion každá položka v poli reprezentuje jeden nebo více znaků, protože příkazy se v tomto případě píší formou textu a ne formou bajtů. Každý konfigurační soubor by v této části měl obsahovat alespoň pět základních struktur. Mezi tyto struktury patří požadavek o číselnou hodnotu, zapsání číselné hodnoty do prvku, zapsání textové hodnoty do prvku, navrácení číselné hodnoty po požadavku a informace o tom, že nastala nějaká událost. Ostatní struktury mohou být přidány pro pozdější funkcionalitu.

První pole hodnot slouží k poslání požadavku o aktuální číselnou hodnotu prvku. U 4D Systems struktury příkazu se tento požadavek skládá z bajtu požadavku čtení hodnoty, bajtu typu objektu, bajtu určující index daného typu objektu a kontrolního součtu.

Druhé pole je k poslání hodnoty do určitého prvku. 4D Systems struktura se skládá ze zapisovacího bajtu, typu objektu, indexu daného typu objektu, dvou bajtů hodnot a kontrolního součtu.

Třetí pole slouží k zapsáni textu do textového prvku. Struktura u značky 4D Systems je bajt pro zapsání textu, bajt indexu textového prvku, bajt počtu znaků, bajty jednotlivých znaků, kde počet musí odpovídat počtu znaků, který byl zadán a nakonec kontrolní součet.

Čtvrté pole slouží k zjištění struktury, která se navrátí po provedení požadavku o hodnotu. Tato struktura obsahuje u 4D Systems bajt, který značí, že jde o vrácení hodnoty po požadavku, bajt nesoucí typ objektu, bajt nesoucí index typu objektu, dva hodnotové bajty a kontrolní součet.

Páté a poslední pole je struktura, která vrací informace o tom, že uživatel provedl nějaký vstup, například klikl na tlačítko. U 4D Systems je struktura téměř identická se strukturou, kde se vrací hodnota po požadavku o ni. Rozdílný je však první bajt. V tomto případě se jedná o bajt, který informuje o události. Zbytek je stejný jako u čtvrtého pole, tedy bajt typu objektu, bajt indexu typu objektu, dva hodnotové bajty a bajt kontrolního součtu.

3.5.4 Čtvrtá část

Čtvrtá část obsahuje jednotlivé prvky, které je možno na displeji zobrazit. Jednotlivé prvky v této části konfiguračního souboru obsahují základní informace o sobě. Řádků je v této části je podstatně více, nicméně se struktury opakují.

První řádek definuje strukturu, která obaluje všechny prvky, které jsou dostupné u daného displeje. V této struktuře se nachází další struktury, které obsahují potřebné informace k jednotlivým prvkům, pro správnou funkci knihovny.

Druhý řádek definuje strukturu daného prvku. V této struktuře se nachází informace o tom, zda je prvek číselný, nebo textový. Dále se zde nachází informace, zda je prvek vstupní, či výstupní. Ještě je potřeba definovat pole použitých prvku daného typu. Do tohoto pole se napíše vždy ID daného prvku. U 4D Systems konfiguračního souboru je také třeba definovat adresu prvku, protože se v příkazech rozlišují i typy objektů bajtem.

Struktura všech prvků by měla být stejná, s rozdílem jednotlivých indexů prvků. Na konci souboru je také důležité nezapomenout uzavřít strukturu elements pro správnou funkčnost.

4 Praktické použití knihovny

Jako nápověda pro použití knihovny je vytvořena ukázková aplikace představující použití vytvořené knihovny a ukazuje funkčnost jednotlivých prvků, jako jsou tlačítka, posuvníky, nebo měřáky. Nejprve bude potřeba nastavit správně Raspberry Pi, poté nastavit všechny prvky na konkrétních displejích, tak ať funguje ukázkový program a nakonec nahrát grafické rozhraní do displeje a spustit program. Pro podrobnější nápovědu je i vytvořena dokumentace, kterou je možno vygenerovat pomocí pydoc.

4.1 Konfigurace Raspberry Pi

V této kapitole se nainstaluje Raspbian na Raspberry Pi, což je distribuce linuxu, která je určena přímo pro Raspberry Pi. Dále se nastaví internetové připojení a nakonec se musí povolit i sériová komunikace, která bude sloužit pro komunikaci se smart displejem.

4.1.1 Instalace Raspbianu

Jako první krok se musí nainstalovat operační systém na Raspberry Pi. K tomu bude potřeba MicroSD karta a PC. Na internetových stránkách Raspberry Pi, pomocí odkazu ke stáhnutí Raspbianu[51], jej stáhneme.

Po stáhnutí Raspbianu bude potřeba ještě nějaký software, který umožní to, že dokáže z .iso obrazu nahrát na externí paměťové zařízení tento obraz, který se po vložení do jiného počítače zavede. Na internetu takových programů je k dispozici spostu. Zde je jako příklad program Rufus[52], což je jednoduchý program tvořený právě pro správné nahrání operačního systému na externí zařízení.

Dále je potřeba připojit MicroSD kartu přes nějaký k adaptér k počítači, na kterém je stažený Raspbian a požadovaný software k jeho správnému uložení na externí zařízení. Po připojení MicroSD karty, je potřeba spustit software a pomocí něj vybrat soubor, který se má zapsat na MicroSD kartu. Dále se vybere výstupní zařízení jako připojenou MicroSD kartu a zahájí zápis.

Až se zapíše operační systém na MicroSD kartu, MicroSD kartu je potřeba vyjmout z počítače a vložit ji do Raspberry Pi. Potom je potřeba připojit monitor k Raspberry Pi přes HDMI, přes USB připojit klávesnici a myš a nakonec připojit microUSB adaptér, pomocí kterého se zapne celé Raspberry Pi.

Po chvíli se na připojeném monitoru zobrazí logo Rasbianu a automaticky se spustí zavádění nového operačního systému. Až se nainstalují všechny potřebné soubory, tak se Raspberry Pi restartuje a spustí se nainstalovaný operační systém Raspbian. Nyní je operační systém nainstalovaný a je možno používat Raspberry Pi jako běžný počítač.

4.1.2 Internetové připojení

Pro připojení Raspberry Pi k internetu existuje několik možností, jak to provést. Pokud je Raspberry Pi verze 3 nebo výše, je možno jej připojit přes zabudovanou Wi-Fi. Toto Raspberry Pi by mělo mít již předinstalované ovladače k svému Wi-Fi modemu. Další možností je využít Ethernetový port a připojit Raspberry Pi pomocí Ethernet kabelu. Tahle možnost bude fungovat i na nižších verzích Raspberry Pi, než je verze 3. Samozřejmě musí to být verze, která má už jako součást Ethernetový port. Problém však nastává, pokud je potřeba připojit Raspberry Pi verzi 2 a nižší k internetu a není k dispozici Ethernetový kabel.

Tento problém se dá vyřešit zakoupením Wi-Fi modemu do USB. Toto by nebyl až takový problém, ale problém je s podporou Raspberry Pi a některých Wi-Fi modemů. Nejjednodušší je zakoupit si modem, který je kompatibilní s Raspberry Pi, protože se ovladače nainstalují samy. Pro modem, který je určený pro desktopové počítače, bude problém s vyhledáním ovladačů pro něj. Existuje však mnoho modemů, tak je potřeba na internetu najít ovladače přesně ke konkrétnímu typu a přenést je na Raspberry Pi, kde se musí nainstalovat. Pokud nám nyní Raspberry Pi identifikuje Wi-Fi modem, je potřeba ještě nakonfigurovat síť, ke které se bude Raspberry Pi připojovat.

Konfigurace sítě na Raspberry Pi

Konfiguraci je možno provést pomocí terminálového okna, nebo pomocí grafického rozhraní. Jelikož je možnost mít verzi Raspbianu bez grafického rozhraní, popsaný způsob bude přes terminálové okno, které mají všechny verze Raspbianu.

Nejprve je potřeba otevřít terminálové okno a zapsat příkaz pro otevření konfiguračního souboru sítí.

```
sudo nano /etc/network/interfaces
```

V terminálovém okně se zobrazí konfigurační soubor, který se upraví tak, že se přidají následující řádky. Pokud se používá šifrování wpa:

```
auto wlan0
iface wlan0 inet dhcp
   wpa-ssid [FSSID]
   wpa-psk [PASSWORD]
```

Pokud se používá šifrování wep

```
auto wlan0
iface wlan0 inet dhcp
   wireless-ssid [FSSID]
   wireless-key [PASSWORD]
```

kde místo [FSSID] patří název Wi-Fi sítě a místo [PASSWORD] se zapíše heslo této Wi-Fi sítě.

Po zapsání příkazů do konfiguračního souboru je potřeba zmáčknout klávesovou zkratku Ctrl+X a poté potvrdit písmenem Y, ať se změny uloží. Teď už by mělo stačit jen restartovat Raspberry Pi, nebo jeho internetovou službu pomocí příkazu.

```
sudo /etc/init.d/networking restart
```

Po restartu by se mělo Raspberry Pi připojit k internetu pomocí našeho Wi-Fi modemu.

4.1.3 Povolení více typů komunikace

Pro povolení různých typů komunikace stačí v desktopovém rozhraní kliknout na ikonu nastavení s názvem Preferences. Poté vybrat Raspberry Pi Configuration a najet do záložky Interfaces. Mělo by se zobrazit okno, kde jsou povoleny a zakázány různé typy komunikací.

V tomto okně je potřeba překliknutím na Enable povolit typy komunikací, které se budou používat. To je SSH a I^2C.

Dále se bude muset povolit UART komunikace. Je potřeba tedy upravit konfigurační soubor */boot/config.txt*. V tomto souboru je potřeba připsat příkaz *enable_uart=1* a uložit jej. Nyní by měla být zajištěná veškerá komunikace, která bude potřeba ze strany Raspberry Pi.

4.2 Praktické nastavení konkrétních displejů

V této kapitole se ukáže, jak se pracuje s programy grafického rozhraní od jednotlivých značek smart displejů. Poté se nastaví jednotlivé prvky tak, aby byl funkční ukázkový program. Nakonec bude předveden postup, jak nahrát grafické rozhraní do displeje.

4.2.1 4D systems

Pro vytvoření grafického rozhraní displeje od firmy 4D systems je třeba stáhnout program 4d workshop na desktopový počítač. Tento program se nachází na stránkách firmy 4D systems[40].

Po spuštění 4d workshop IDE se musí zvolit displej s jakým se bude pracovat z nabídky a jestli bude používáno vertikální nebo horizontální rozložení. Rozložení se mění pomocí klikání na obrázek displeje. V tomto případě se zvolí uLCD-32PTU a horizontální rozložení. Poté je možnost vybrat ze 4 typů prostředí. Designer, ViSi, ViSi Genie a Serial. Vybere se možnost ViSi Genie a po kliknutí se otevře nové prostředí.

V tomto prostředí se nachází pravá horní nabídka, kde jsou různé prvky, jako například tlačítka, kruhové ukazatele hodnot, přepínače nebo posuvníky. Každý prvek je možno jednoduše přetáhnout na obrázek displeje a umístit ho tak, jak je požadováno aby se zobrazoval později.

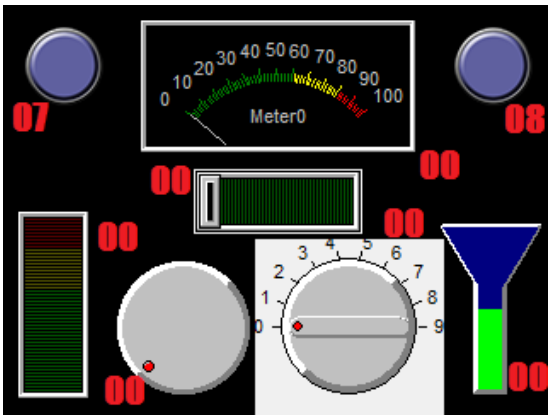
Jakmile se přetáhne a umístí nějaký prvek, tak se dole vpravo zobrazí vlastnosti daného prvku. Vlastnostmi prvku může být velikost, pozice, barva, počáteční hodnota, a tak dále. Také je možnost si přidat novou obrazovku, takže není omezeni pouze na jednu obrazovku.

Pokud se umístí na displej prvek, který má funkci vstupu, je potřeba mu nastavit událost. Přejde se ze záložky *Properties* do záložky *Events* a na událost *OnChanged* se přidá handler *Report message*. Tato událost zajistí, že vstupní prvek po stisknutí nebo po změně hodnoty pošle informaci na sériovou linku.

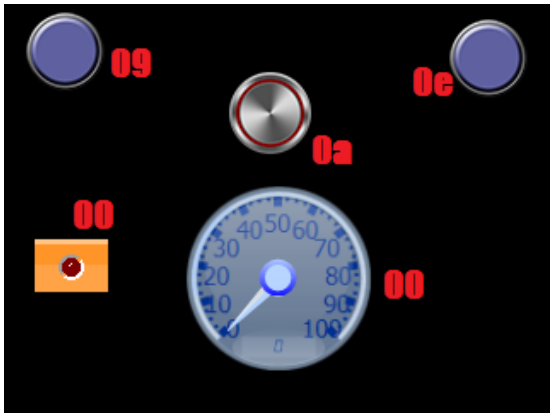
V záložce *Properties* má každý prvek své jedinečné jméno, podle kterého je možno přes sériovou linku požádat o hodnotu nebo hodnotu přiřadit. První část jména je název prvku, který je poté v sériové komunikaci nahrazen hexadecimálním číslem daného prvku. Druhá část je číslo, což je index, který značí pořadové číslo stejných prvků.

Nastavení prvků displeje pro ukázkový program

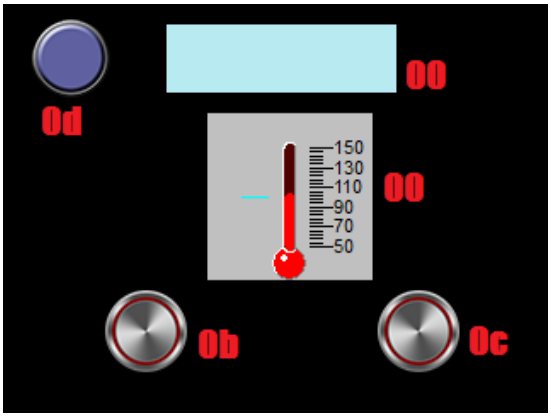
(a) První obrazovka



(b) Druhá obrazovka



(c) Třetí obrazovka



(d) Čtvrtá obrazovka

Obrázek 16: Jednotlivé obrazovky pro ukázkový program

Pro správnou funkčnost ukázkového programu je potřeba vytvořit k němu odpovídající grafické rozhraní. Ukázkový program obsahuje 4 obrazovky, které se musí patřičně nastavit. Obrazovky jsou vyobrazeny na obrázcích 16a, 16b, 16c a 16d. Na obrazovkách u konkrétních prvků je

vždy vyobrazeno číslo. Toto číslo je index daného prvku, který musí být dodržený pro správnou funkčnost ukázkového programu.

U tlačítek s indexy 06, 07, 08, 09, 0e a 0d musí být nastavena událost, která změní aktivní obrazovku po kliknutí na dané tlačítko. Tyto tlačítka jsou umístěny v horních rozích a představují plynulý přechod mezi obrazovkami. U tlačítek se nastaví události přechodu tak, že se označí dané tlačítko a v záložce *Events* se nastaví událost na *FormXActivate*, kde X je číslo dané obrazovky. Nastaví se tedy tlačítku s indexem 06 přechod na obrazovku form1, 07 na form0, 08 na form2, 09 na form1, 0e na form3 a 0d na form2. U prvků není důležité umístění, takže mohou být jinak uspořádány.

Až se rozmístí a nastaví všechny prvky, které mají být na obrazovkách, uloží se soubor. Dále bude potřeba propojit displej s počítačem a nahrát do něj grafické rozhraní, které bylo vytvořeno.

Propojení a nahrání grafického rozhraní

K propojení 4D systems displeje a počítače bude potřeba převodník z UART na microUSB. Převodník je součástí původního balení k displeji. Jako další věc bude potřeba kabel, kterým se propojí desktopový počítač s microUSB a 5-pinový kabel, který by měl být taktéž součástí původního balení k displeji.

Nejprve se připojí 5-pinový kabel k zadní části displeje, kde je potřeba najít označení pro UART, což je série pinů +5V, TX, RX, GND, RES. Druhá strana 5-pinového kabelu se připojí k zmiňované součástce. Je potřeba si ověřit zda jdou všechny kabely do součástky tak, jak vycházejí z displeje s výjimkou RX a TX, které se prohazují. Nakonec se připojí microUSB kabel z jedné strany do součástky a z druhé strany do počítače. Nyní by se měl displej rozsvítit. Pokud to tak je, zapojení bylo správné.

Pro nahrání grafického rozhraní do displeje se nejprve spustí 4d workshop. Dále se vytvoří nový, nebo se otevře již uložený projekt, který se bude do displeje nahrávat. V horní liště se přejde na záložku *Comms*. V této záložce se vybere *COM*, pod kterým je připojený displej a klepne se na červenou kuličku. Pokud se komunikace spáruje, vše se úspěšně nastavilo a může se nahrát do displeje grafické rozhraní. Přejde se zpět do záložky Home a ikonou s nápisem *Build Copy/Load* zahájíme sestavení programu. Program se zkompiluje a grafické rozhraní se nahraje do displeje přes připojený port. Jakmile vše proběhne, mělo by jít vidět na displeji grafické rozhraní, které se nastavovalo v programu a posílalo na displej.

4.2.2 Nextion

Pro vytvoření grafického rozhraní na displeji od firmy Nextion, je třeba si stáhnout program Nextion Editor[7] z webových stránek.

Jakmile se spustí program Nextion Editor, klikne se na nový projekt a otevře se okno se základním nastavením displeje. Zde se musí vybrat jaký displej se používá a jaké rozložení se požaduje. Jestli má být displej pootočený o 0, 90, 180, 270 stupňů. V tomto případě se zvolí displej NX3224K024_011 a jako pootočení se zvolí 270 stupňů, ať má orientaci na šířku. Jakmile

se potvrdí výběr, naběhne uprostřed rámeček, do kterého se mohou vkládat prvky displeje. Prvky, které se mohou do okénka přetáhnout, se nachází vlevo v rámečku *Toolbox*. Pro úpravu obrazovek slouží rámeček vpravo.

Pro umístění prvku na displej se konkrétní prvek vyhledá v okně *Toolbox* a klikne se na něj. Po kliknutí se prvek zobrazí v bílém rámečku uprostřed, který reprezentuje obrazovku displeje. Prvek je možno přetáhnutím posunout na libovolnou pozici nebo jej zvětšit či zmenšit. Pokud je prvek nakliklý, tak se vpravo dole zobrazují vlastnosti daného prvku, které je možno sledovat nebo upravovat.

Prvky displeje od firmy Nextion se ovládají přes vlastnost *objname*, takže je třeba vědět, který prvek je jak označený. Toto se musí zajistit manuálně přes zprávy událostí, pokud bude potřeba vědět vědět, na které tlačítko uživatel kliknul. V dolní části obrazovky je okno s názvem *Events*. V tomto okně jsou dvě záložky a z nich se přejde do záložky *Touch Release Event*. V této záložce je potřeba zajistit, že když uživatel klikne na tlačítko nebo změní hodnotu prvku dotykem, aby displej poslal zprávu o události. To se zajistí tím, že se pošle *objname* skrze sériovou linku a pokud bude prvek mít i nějaký stav, tak i jeho hodnotu. Do záložky *Touch Release Event* se tedy napíše příkaz *prints* a do uvozovek název *objname*. Jako druhý parametr je počet bajtů, které mají být poslány. Pokud se zadá 0, pošlou se automaticky všechny bajty. Parametry se oddělí následovně.

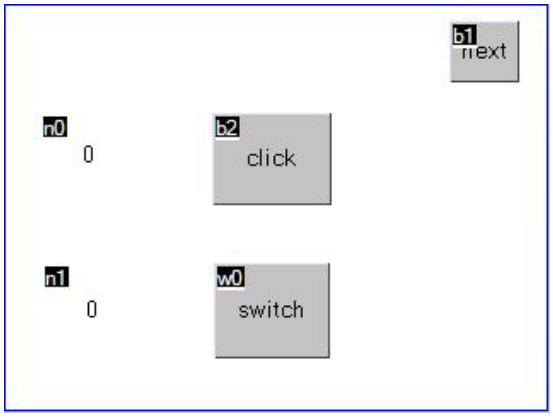
```
prints "b0",0
```

Pokud se jedná o prvek, který má i nějakou hodnotu připíše se pod příkaz prints další příkaz prints <objname>.val. Informace o události u takového prvku by tedy mohla vypadat následovně.

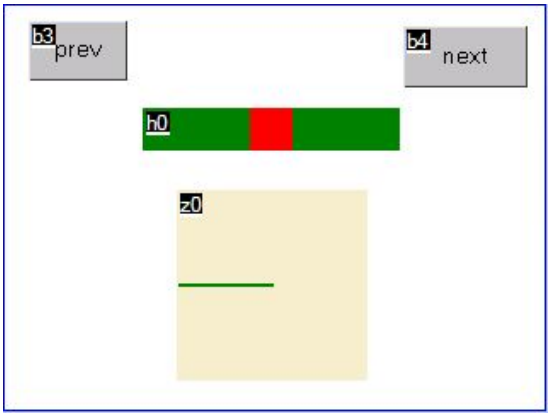
```
prints z0.val,0
```

Dohromady se tedy bude jednat o dva příkazy.

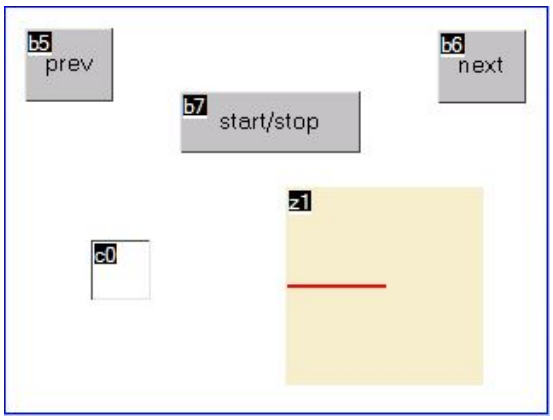
Nastavení prvků displeje pro ukázkový program



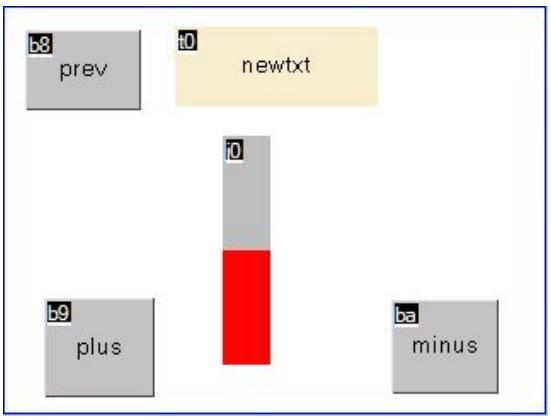
(a) První obrazovka



(b) Druhá obrazovka



(c) Třetí obrazovka



(d) Čtvrtá obrazovka

Obrázek 17: Jednotlivé obrazovky pro ukázkový program

Pro funkčnost ukázkového programu je třeba vytvořit prvky podle čtyř obrazovek na obrázcích 17a, 17b, 17c a 17d. U prvků je důležité dodržet správné identifikátory, protože by jinak ukázkový program nefungoval správně. Dále je důležité nastavit tlačítkům s textem next a prev událost, která změní obrazovku. V Nextion Editoru se musí označit tlačítko a v dolní části se objeví *Touch release eventy*. Do těchto eventů je potřeba napsat příkaz *page pageX*, kde X je číslo obrazovky. Takže tlačítko b1 bude mít událost v *Touch release eventech* na page1, b3 na page0, b4 na page2, b5 na page1, b6 na page3 a b8 na page2. Prvky je možno umístit libovolně, protože to nemá vliv na funkčnost programu.

Až se umístí všechny prvky tak, jak je požadováno, může se program zkompilovat. Pro kompilaci se stiskne tlačítko *Compile* v horní liště programu. Dole by se měl zobrazit výstup, zda kompilace proběhla úspěšně.

Nahrání grafického rozhraní

Pro nahrání grafického rozhraní do displeje bude potřeba microSD karta. Tato microSD karta

se zapojí do počítače, kde je program Nextion Editor. Dále se otevře program Nextion Editor a otevře se projekt, který je potřeba nahrát do displeje. Až se projekt načte, klikne se na tlačítko *File* v horní liště a vybere se *TFT file output*. Poté se zobrazí okno, kde se vybere výstup zkompilovaného projektu. Vybere se tedy microSD karta, která by neměla obsahovat jiný soubor, než ten, který se bude nahrávat.

Po nahrání projektu na microSD kartu, se karta vyjme z počítače a vloží se do adaptéru na displeji. Displeji se přivede napětí z jakéhokoliv zdroje a displej by se měl rozsvítit s nápisem, že započne ukládání z microSD karty do paměti displeje. Až se zobrazí nápis, že nahrání do displeje bylo úspěšné, tak se odpojí napájení a vyjme se microSD karta. Po restartu displeje, by měla být vidět obrazovka, která se navrhovala v programu.

4.2.3 Ukázková aplikace

Po nastavení všech kroků se musí připojit smart displej k Raspberry Pi. Oba displeje, 4D Systems i Nextion, používají ke komunikaci UART. K Raspberry Pi se displej připojí pomocí pinů s označením 4, 6, 8 a 10. Pin s označením 4 je 5V napětí, 6 je uzemnění, 8 je UART vysílač a 10 je UART přijímač. Jednotlivé piny se tedy připojí k odpovíadjícím na displeji, s vyjímkou vysílače a přijímače, které se mezi sebou prohazují, tedy vysílač displeje půjde na přijímač Raspberry Pi a přijímač displeje půjde na vysílač Raspberry Pi. Pokud je Raspberry Pi přivedeno k napětí, měl by se displej rozsvítit i s předem nastaveným grafickým rozhraním.

Dále je potřeba mít na Raspberry Pi přítomnou knihovnu, která se nachází v příloze bakalářské práce. Tato knihovna se může nahrát na Raspberry Pi manuálně, nebo pomocí SSH z jiného počítače. Ve složce knihovny se nachází složka s konfiguračními soubory, složka s jednotlivými zdrojovými kódy knihovny a dva soubory, 4dDemo.py a nextionDemo.py. Pomocí SSH, nebo manuálního připojení k Raspberry Pi, je potřeba spustit patřičný soubor, podle toho jaký displej je připojený. Pokud je připojený displej od 4D Systems spustí se soubor 4dDemo.py a pokud je připojený Nextion displej spustí se soubor nextionDemo.py. Jakmile se patřičný soubor spustí, mohou se začít testovat různé prvky, které se nachází na displeji. Po kliknutí na prvek se do konzoly vypíše název objektu a jeho hodnota. V ukázkové aplikaci je znázorněno, jak se pomocí jednotlivých vstupních prvků ovládají výstupní prvky.

5 Závěr

Cílem této práce bylo vytvořit knihovnu, která má pomáhat programátorům při vyvíjení programů ke smart displejům. Knihovna tedy zjednodušuje práci programátorům, protože mohou použít jeden program pro více displejů nebo více typů komunikace. Komunikace je sjednocena pro značky displejů 4D Systems a Nextion, přičemž oba displeje komunikují pomocí UARTu. Knihovna nejen sjednocuje komunikaci více značek displejů nebo komunikačních rozhraní, ale i zjednodušuje celkovou komunikaci, protože pomocí jednoho příkazu je možné zapsat hodnotu na displej, nebo tuto hodnotu přečíst. Implementací a otestováním knihovny bylo dosaženo cílů této práce.

Při řešení této práce bylo potřeba překonat celou řadu obtíží, jako je například nastavit u Raspberry Pi bezdrátové připojení k internetu, jelikož nemá zabudovanou Wi-Fi síťovou kartu. Obtížné bylo také nastavit komunikaci mezi běžným počítačem a Raspberry Pi skrze SSH bez nutnosti zadávat heslo. Další výzvou bylo zjistit, v jakém formátu požadují jednotlivé smart displeje komunikační zprávy, jelikož smart displej značky 4D Systems požaduje zakódované číselné hodnoty a smart displej značky Nextion požaduje příkazy zakódované do bajtového řetězce, je přístup velmi rozdílný. Nemalé potíže činilo i zachycení událostí, protože některé události kolidovaly s jinými příkazy, které se posílaly přes sériovou linku a pokud se zde nastavil časový rozestup mezi zprávami, tak byla komunikace příliš opožděná.

Knihovna je vytvořena tak, aby se mohla postupně rozšiřovat. Jelikož je zajištěna komunikace pouze pro displeje značek 4D Systems a Nextion a komunikační protokol UART, je potřeba ji v budoucnu rozšířit o více značek displejů a doplnit implementaci komunikačních prokolů I^2C a SPI. Další značky displejů se přidávají do knihovny pomocí vytvoření nového konfiguračního souboru a vytvoření tříd, které se starají o kódování a dekódování komunikačních zpráv. Komunikační protokoly I^2C a SPI jsou již předpřipravené, takže stačí doplnit implementaci. Pro podrobnější popis jednotlivých tříd je možno vygenerovat dokumentaci pomocí pydoc.

Literatura

1. *Medium: Smart zařízení a propojená zařízení* [online]. 2017 [cit. 2020-04-10]. Dostupné z: https://medium.com/all-technology-feeds/smart-connected-and-iot-based-devices-whats-the-difference-36fc1bdc36b2.
2. *Very Possible: Smart zařízení a propojená zařízení* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.verypossible.com/blog/smart-vs-connected-products-whats-the-difference.
3. *Rutronik: Inteligentní displeje* [online]. 2020 [cit. 2020-04-18]. Dostupné z: https://www.rutronik.com/article/detail/News/intelligent-displays-getting-to-market-faster-with-modules/.
4. *Inductive Automation: Co je to HMI* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.inductiveautomation.com/resources/article/what-is-hmi.
5. *4D Systems uLCD-32PTU: Obrázek produktu uLCD-32PTU* [online]. 2020 [cit. 2020-04-19]. Dostupné z: https://4dsystems.com.au/products/4d-intelligent-hmi-display-modules/microlcd-display-modules/ulcd-32ptu.
6. *4D Systems: Produkty značky 4D Systems* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://4dsystems.com.au/products/featured-products.
7. *Nextion: Produkty značky Nextion* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://nextion.tech/.
8. *Nextion NX4832K035: Obrázek produktu NX4832K035* [online]. 2020 [cit. 2020-04-19]. Dostupné z: https://cdn.nextion.tech/wp-content/uploads/2017/07/Nextion-Enhanced-Generic-3.5-HMI-Touch-Display-2.jpg.
9. *Nextion Basic: Produkty značky Nextion typu Basic* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://nextion.tech/basic-series-introduction/.
10. *Nextion Enhanced: Produkty značky Nextion typu Enhanced* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://nextion.tech/enhanced-series-introduction/.
11. *Nextion Intelligent: Produkty značky Nextion typu Intelligent* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://nextion.tech/intelligent-series-introduction/.
12. *MikroElektronika 3.5": Specifikace produktu Mikroelektronika 3.5"* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://www.mikroe.com/mikromedia-hmi-35-no-touch.
13. *MikroElektronika HMI 3.5": Obrázek produktu HMI 3.5"* [online]. 2020 [cit. 2020-04-19]. Dostupné z: https://www.mikroe.com/mikromedia-hmi-35-no-touch.
14. *MikroElektronika 4.3": Specifikace produktu Mikroelektronika 4.3"* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://www.mikroe.com/mikromedia-hmi-43-no-touch.

15. *MikroElektronika 5.0": Specifikace produktu Mikroelektronika 5.0"* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://www.mikroe.com/mikromedia-hmi-50-no-touch.

16. *MikroElektronika 7.0": Specifikace produktu Mikroelektronika 7.0"* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://www.mikroe.com/mikromedia-hmi-70-no-touch.

17. *Electronic Assembly 2.8": Dokumentace produktu Electronic Assembly 2.8"* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.lcd-module.com/fileadmin/eng/pdf/grafik/edip128-6e.pdf.

18. *Electronic Assembly 3.2": Dokumentace produktu Electronic Assembly 3.2"* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.lcd-module.com/fileadmin/eng/pdf/grafik/edip160-7e.pdf.

19. *Electronic Assembly 4.2": Dokumentace produktu Electronic Assembly 4.2"* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.lcd-module.com/fileadmin/eng/pdf/grafik/edip240-7e.pdf.

20. *Electronic Assembly 5.7": Dokumentace produktu Electronic Assembly 5.7"* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.lcd-module.com/fileadmin/eng/pdf/grafik/edip320-8e.pdf.

21. *Riverdi: Výpis produktů značky Riverdi* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://riverdi.com/product-category/eve/.

22. *Riverdi RiTFT-35-CAP: Obrázek produktu RiTFT-35-CAP* [online]. 2020 [cit. 2020-04-19]. Dostupné z: https://riverdi.com/wp-content/uploads/2019/01/35inch-eve3-galleryArtboard-1-copy-8.png.

23. *Wikipedia: Raspberry Pi* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi.

24. *Tux graphics GPIO: Rozložení GPIO pinů na Raspberry Pi 2 Model B* [online]. 2020 [cit. 2020-04-13]. Dostupné z: http://tuxgraphics.org/npa/raspberry-pi-gpio-101/gpio_layout-raspberry-pi2.jpg.

25. *The pin hut: Introduction to Raspberry Pi GPIO* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://thepihut.com/blogs/raspberry-pi-tutorials/an-introduction-to-raspberry-pi-gpio.

26. *Pinout: Description of Raspberry Pi pins* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://pinout.xyz/.

27. *Wikipedia Raspberry Pi architecture: Architektura Raspberry Pi 2 Model B* [online]. 2020 [cit. 2020-04-13]. Dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi#/media/File:Raspberry_Pi_B+_rev_1.2.svg.

28. *Fossmint: Raspberry Pi operating systems by Fossmint* [online]. 2020 [cit. 2020-05-04]. Dostupné z: https://www.fossmint.com/operating-systems-for-raspberry-pi/.

29. *Sparkfun: Popis sériové komunikace* [online]. 2019 [cit. 2020-04-10]. Dostupné z: https://learn.sparkfun.com/tutorials/serial-communication/all.

30. *Codrey: Co je sériová komunikace* [online]. 2018 [cit. 2020-04-10]. Dostupné z: https://www.codrey.com/embedded-systems/serial-communication-basics/.

31. *Technopedia: Transistor transistor logic* [online]. 2016 [cit. 2020-04-10]. Dostupné z: https://www.techopedia.com/definition/3057/transistor-transistor-logic-ttl.

32. *UART Wikipedia: Popis UART* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

33. *RS-232 Wikipedia: Popis RS-232* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/RS-232.

34. *TTL Wikipedia: Popis TTL* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic.

35. *I2C-Bus: Adresování v I2C* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://www.i2c-bus.org/addressing/.

36. *I2C Wikipedia: Popis I2C* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/I%C2%B2C.

37. *SPI Wikipedia: Popis SPI* [online]. 2020 [cit. 2020-04-10]. Dostupné z: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

38. *Independent slave configuration: Obrázek zapojení independent slave konfigurace* [online]. 2006 [cit. 2020-04-19]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/f/fc/SPI_three_slaves.svg.

39. *Daisy chain configuration: Obrázek zapojení daisy chain konfigurace* [online]. 2006 [cit. 2020-04-19]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/9/97/SPI_three_slaves_daisy_chained.svg.

40. *4D Systems: Webové stránky značky 4D Systems* [online]. 2020 [cit. 2020-04-12]. Dostupné z: https://4dsystems.com.au/.

41. *4D Systems uLCD-32PTU: Dokumentace 4D Systems uLCD-32PTU* [online]. 2019 [cit. 2020-04-10]. Dostupné z: https://4dsystems.com.au/mwdownloads/download/link/id/349/.

42. *4D Systems ViSi-Genie: Popis programu ViSi-Genie od 4D Systems* [online]. 2019 [cit. 2020-04-10]. Dostupné z: https://4dsystems.com.au/mwdownloads/download/link/id/26/.

43. *Nextion: Webové stránky značky Nextion* [online]. 2020 [cit. 2020-04-12]. Dostupné z: https://nextion.tech/.

44. *Digikey nx3224k024: Dokumentace displeje NX3224K024* [online]. 2020 [cit. 2020-04-10]. Dostupné z: `https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/104060028_Web.pdf`.

45. *Nextion instruction set: Popis instrukcí displeje Nextion* [online]. 2020 [cit. 2020-04-10]. Dostupné z: `https://nextion.tech/instruction-set/`.

46. *Surenoo: Webové stránky značky Surenoo* [online]. 2020 [cit. 2020-04-12]. Dostupné z: `https://www.surenoo.com/`.

47. *H28A-I SPEC: Basic information about display Surenoo H28A-I.* martin$_x$*hli*@126.*com*, 2018.

48. *vLcds I2C Instruction set: Instructions used in vLcds I2C displays.* Surenoo, 2018.

49. *Visual Lcd Studio: User Manual of Visual Lcd Studio.* Hunda Tech, 2018.

50. *Visual Studio Code: Stáhnutí programu VS Code* [online]. 2020 [cit. 2020-04-29]. Dostupné z: `https://code.visualstudio.com/`.

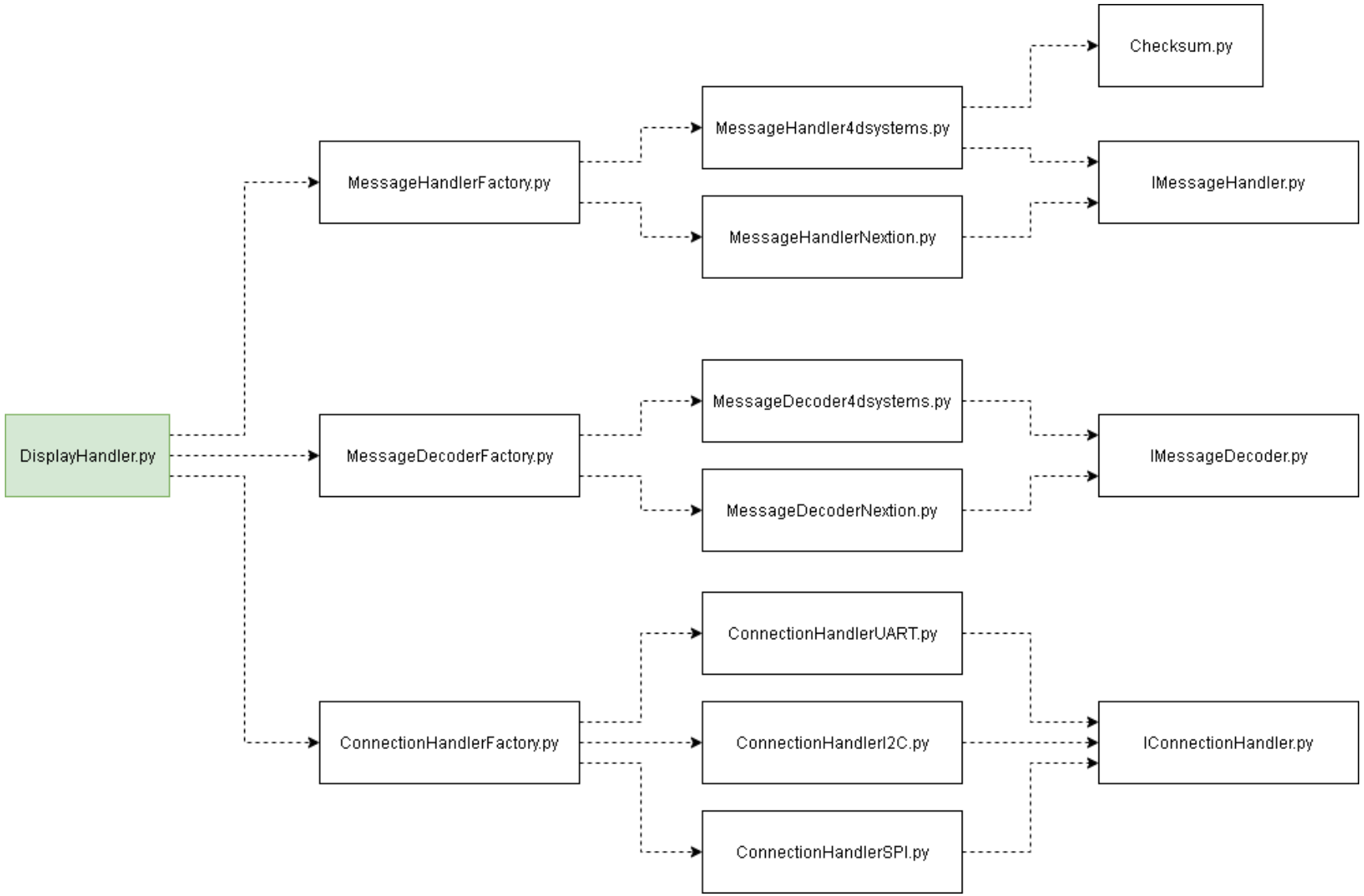
51. *Raspberry Pi Raspbian: Stažení operačního systému Raspbian* [online]. 2020 [cit. 2020-04-11]. Dostupné z: `https://www.raspberrypi.org/downloads/`.

52. *Rufus: Program k vytvoření bootovacího USB* [online]. 2020 [cit. 2020-04-11]. Dostupné z: `https://rufus.ie/`.

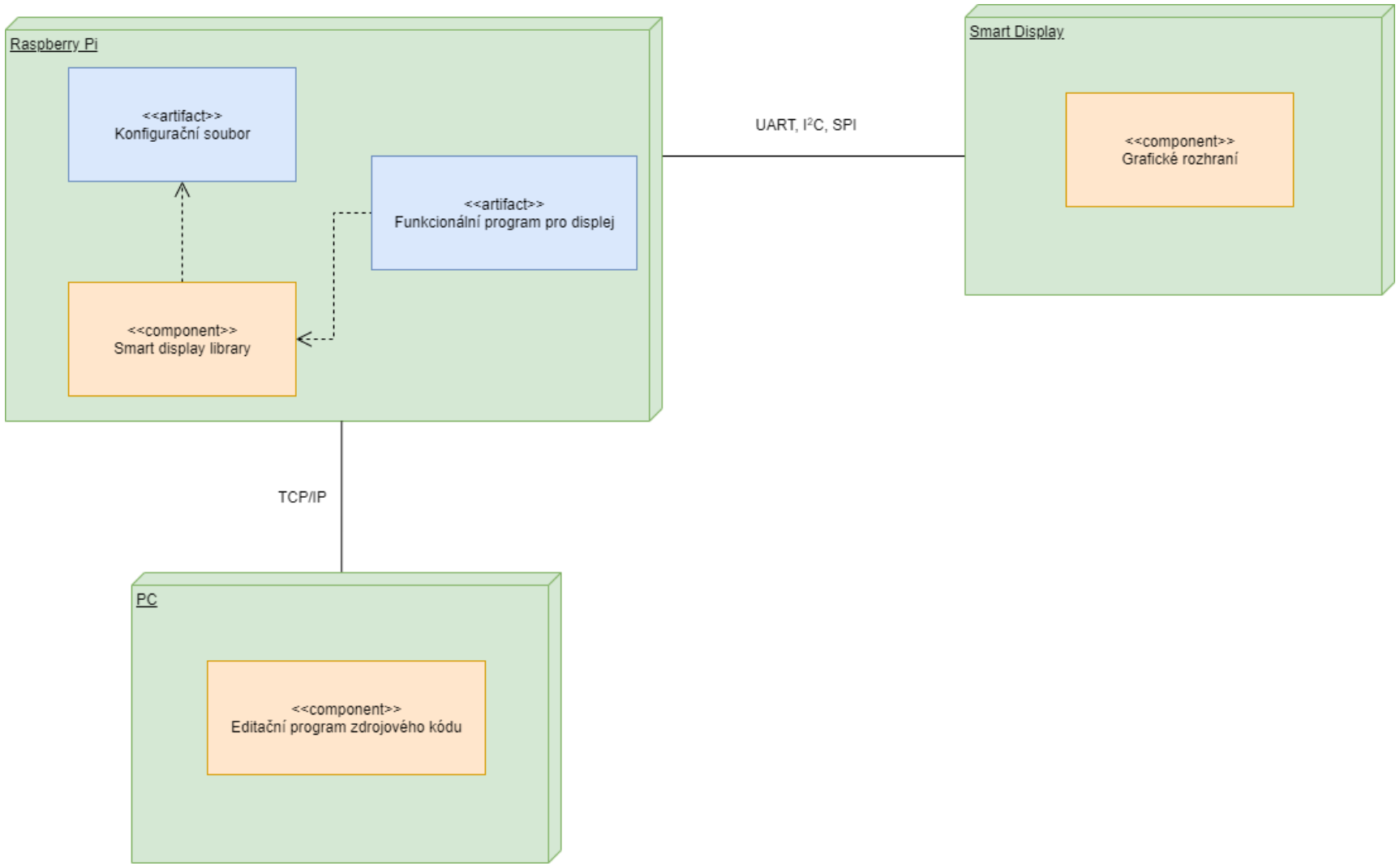
A Tabulky a obrázky

Tabulka 6: Jednotlivé displeje značky 4D Systems

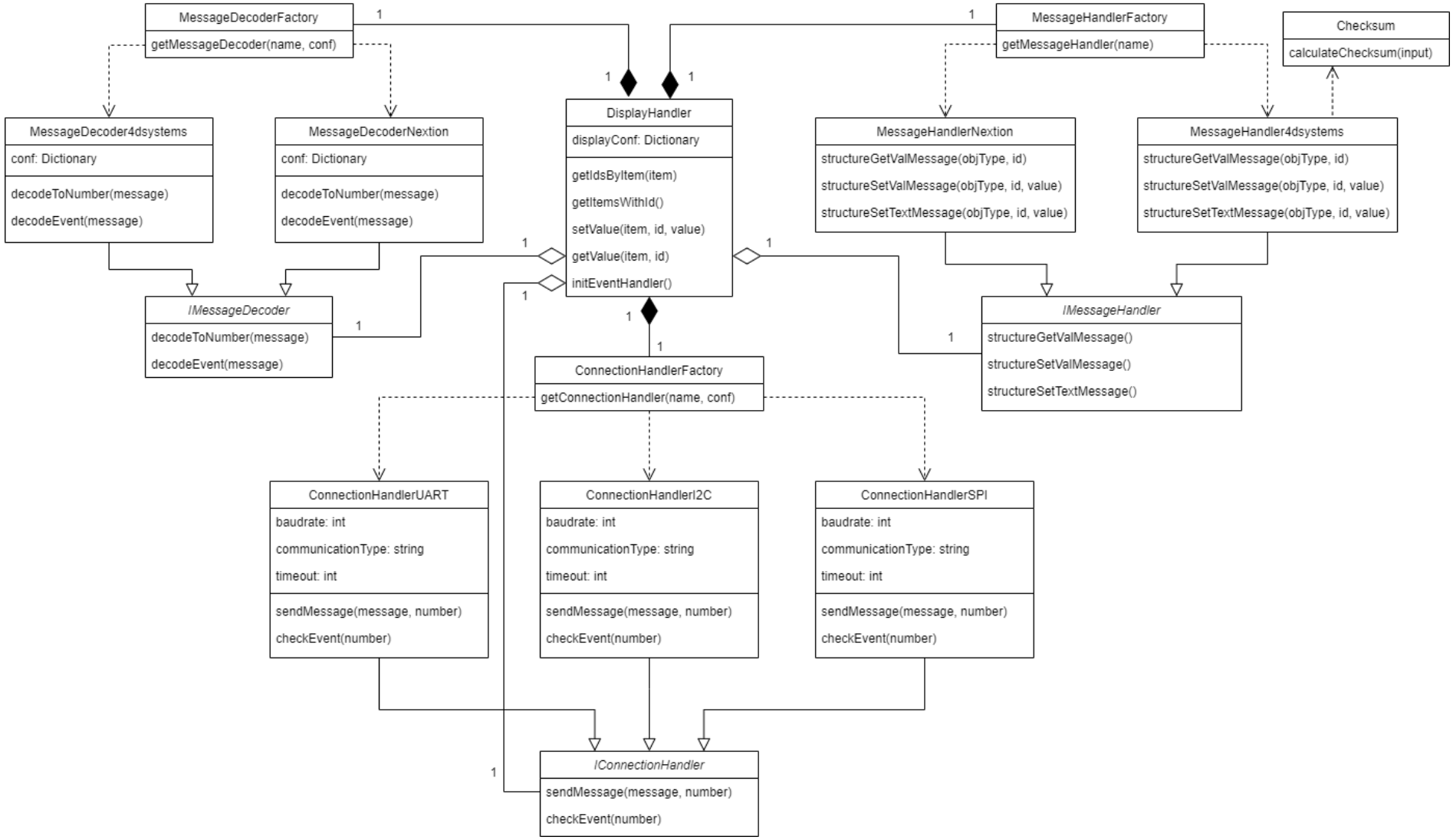
Název	Velikost	Rozlišení	Typ dotyku	Rozhraní	Cena
ULCD-144-G2	1,44 palců	128×128 pixelů	Žádný	UART	35 USD
ULCD-220RD	1,38 palců	220×220 pixelů	Žádný	UART	69 USD
ULCD-24PTU	2,4 palců	240×320 pixelů	Rezistivní	UART	59 USD
ULCD-28PTU	2,8 palců	240×320 pixelů	Rezistivní	UART	69 USD
ULCD-32PTU	3,2 palců	240×320 pixelů	Rezistivní	UART	79 USD
ULCD-35DT	3,5 palců	320×480 pixelů	Rezistivní	UART	89 USD
ULCD-43DT	4,3 palců	480×272 pixelů	Rezistivní	UART	139 USD
ULCD-70DT	7,0 palců	800×480 pixelů	Rezistivní	UART	179 USD
ULCD-90DT	9,0 palců	800×480 pixelů	Rezistivní	UART	186 USD
ULCD-90DCT	9,0 palců	800×480 pixelů	Kapacitní	UART	220 USD
UOLED-96-G2	0,96 palců	96×64 pixelů	Žádný	UART	45 USD
UOLED-128-G2	1,5 palců	128×128 pixelů	Žádný	UART	55 USD
UOLED-160-G2	1,7 palců	160×128 pixelů	Žádný	UART	65 USD



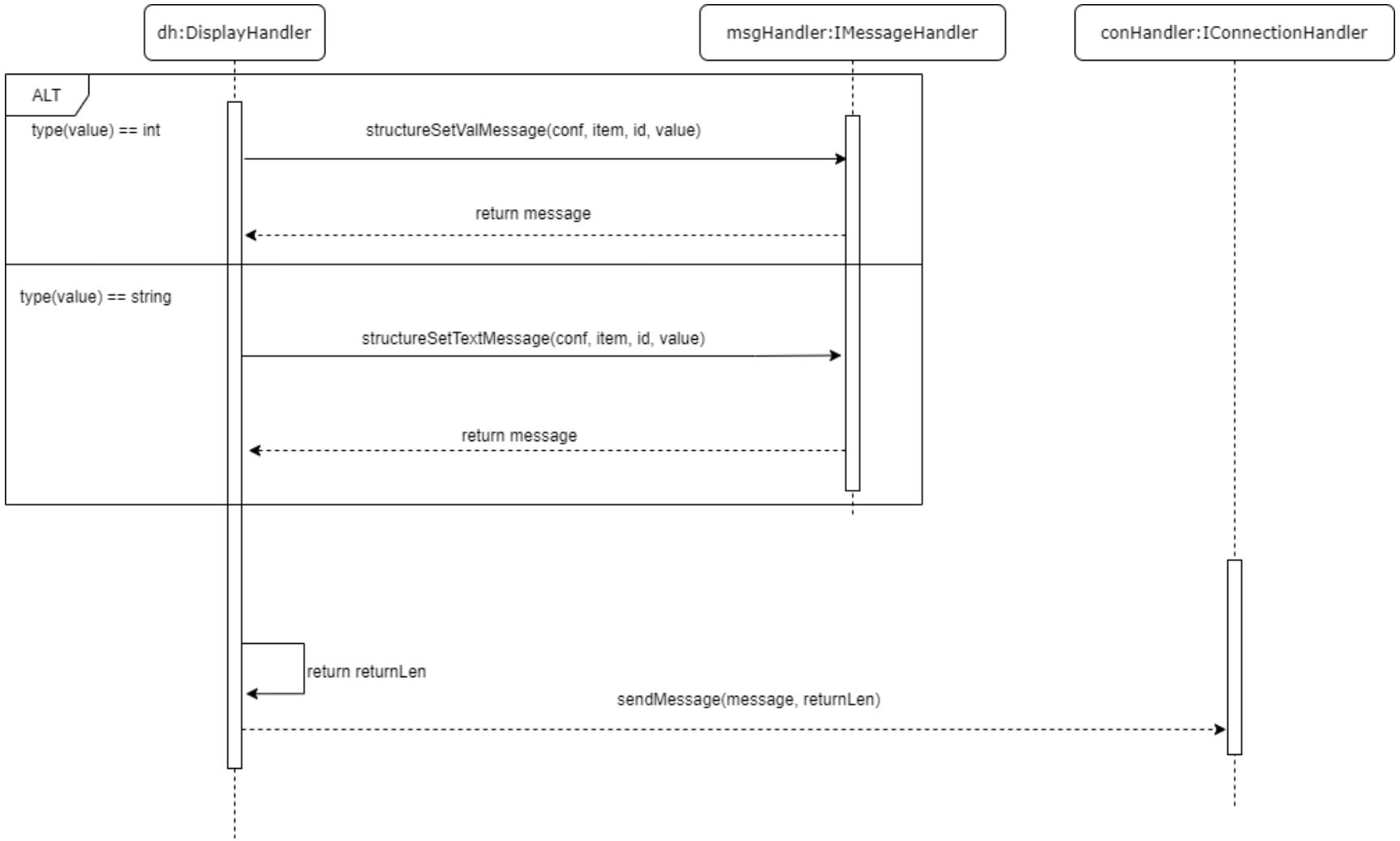
Obrázek 18: Diagram popisující závislosti mezi soubory



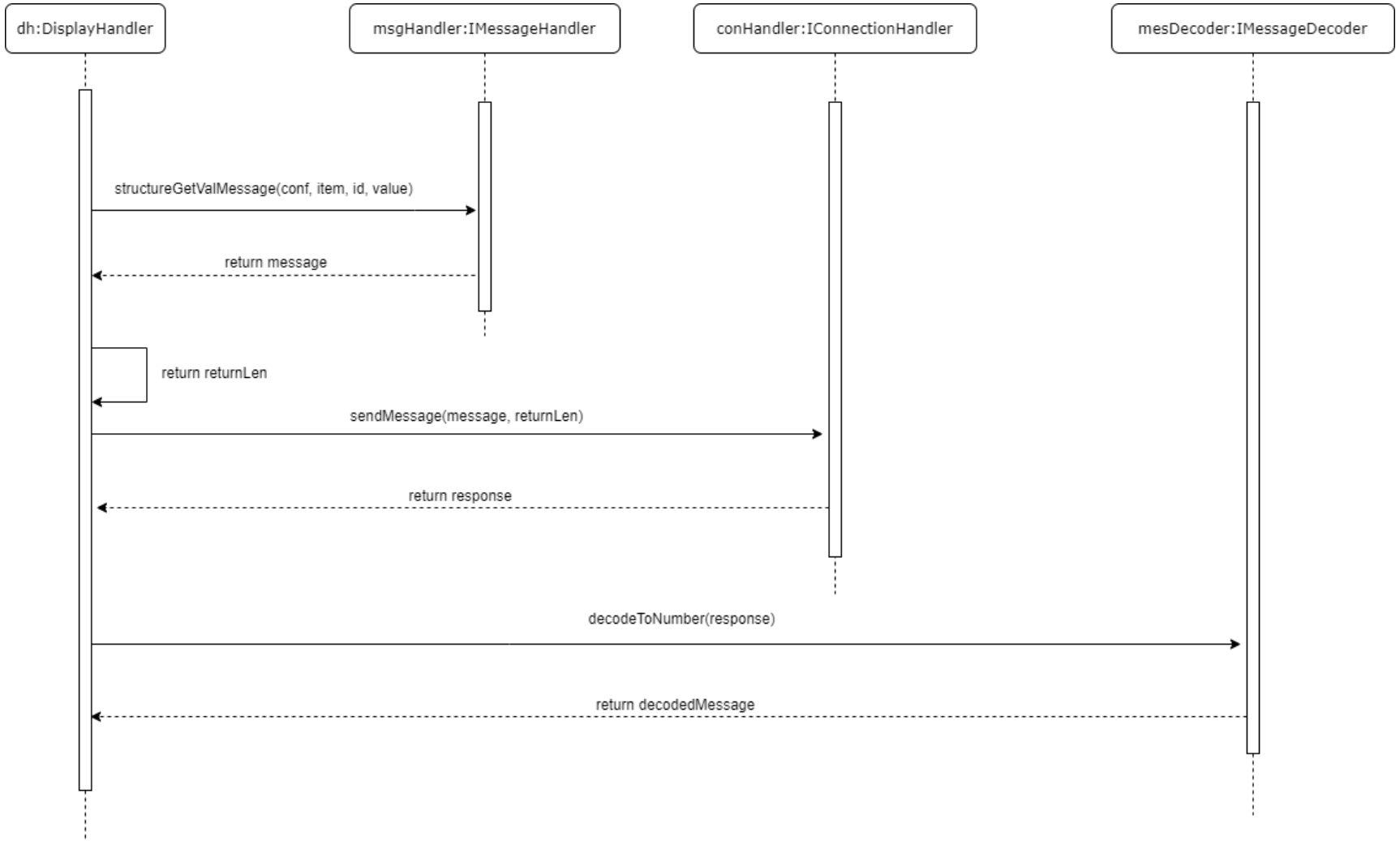
Obrázek 19: Diagram nasazení



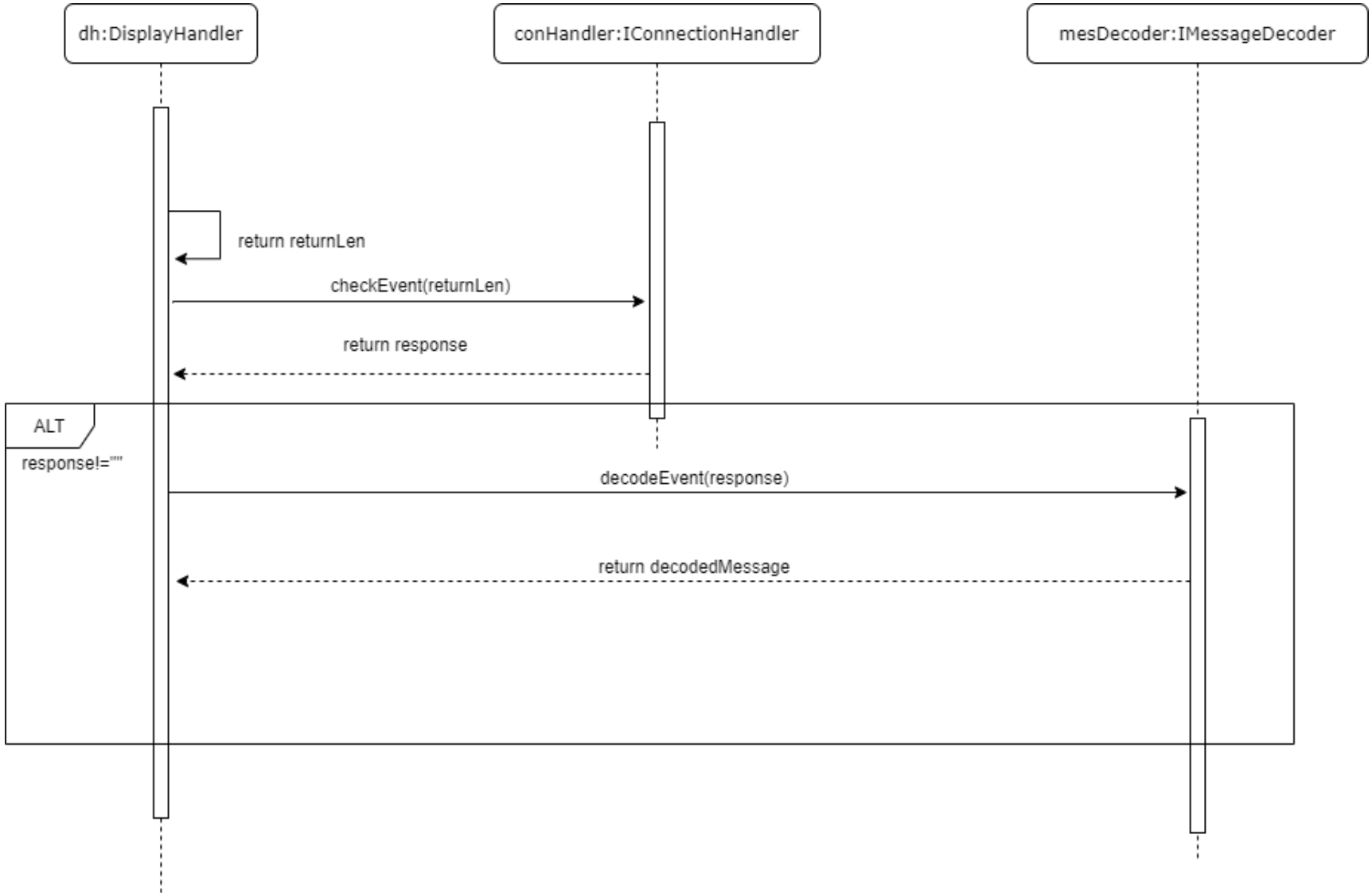
Obrázek 20: Třídy v knihovně a jejich vzájemné vazby



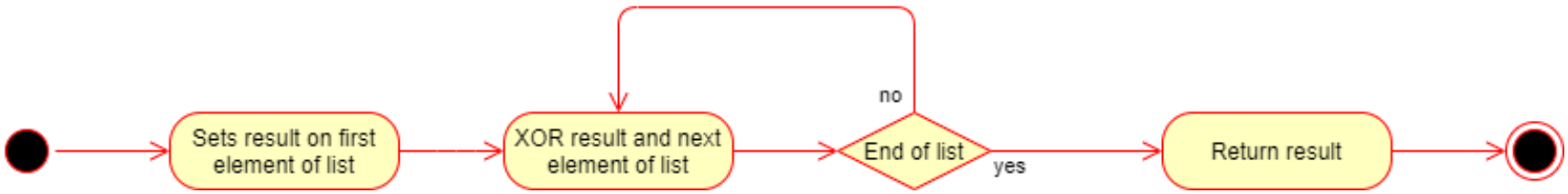
Obrázek 21: Sekvenční diagram metody setValue



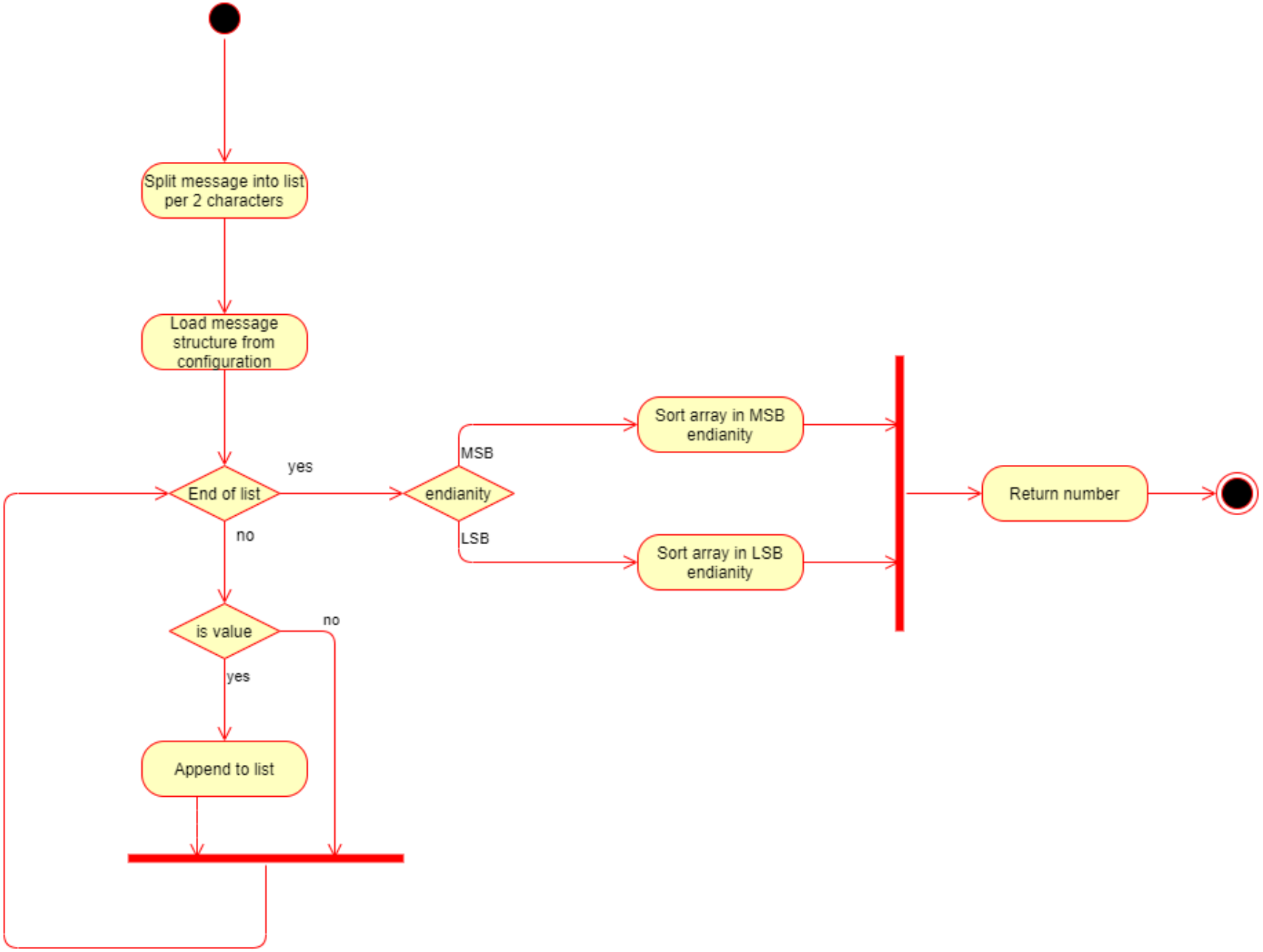
Obrázek 22: Sekvenční diagram metody getValue



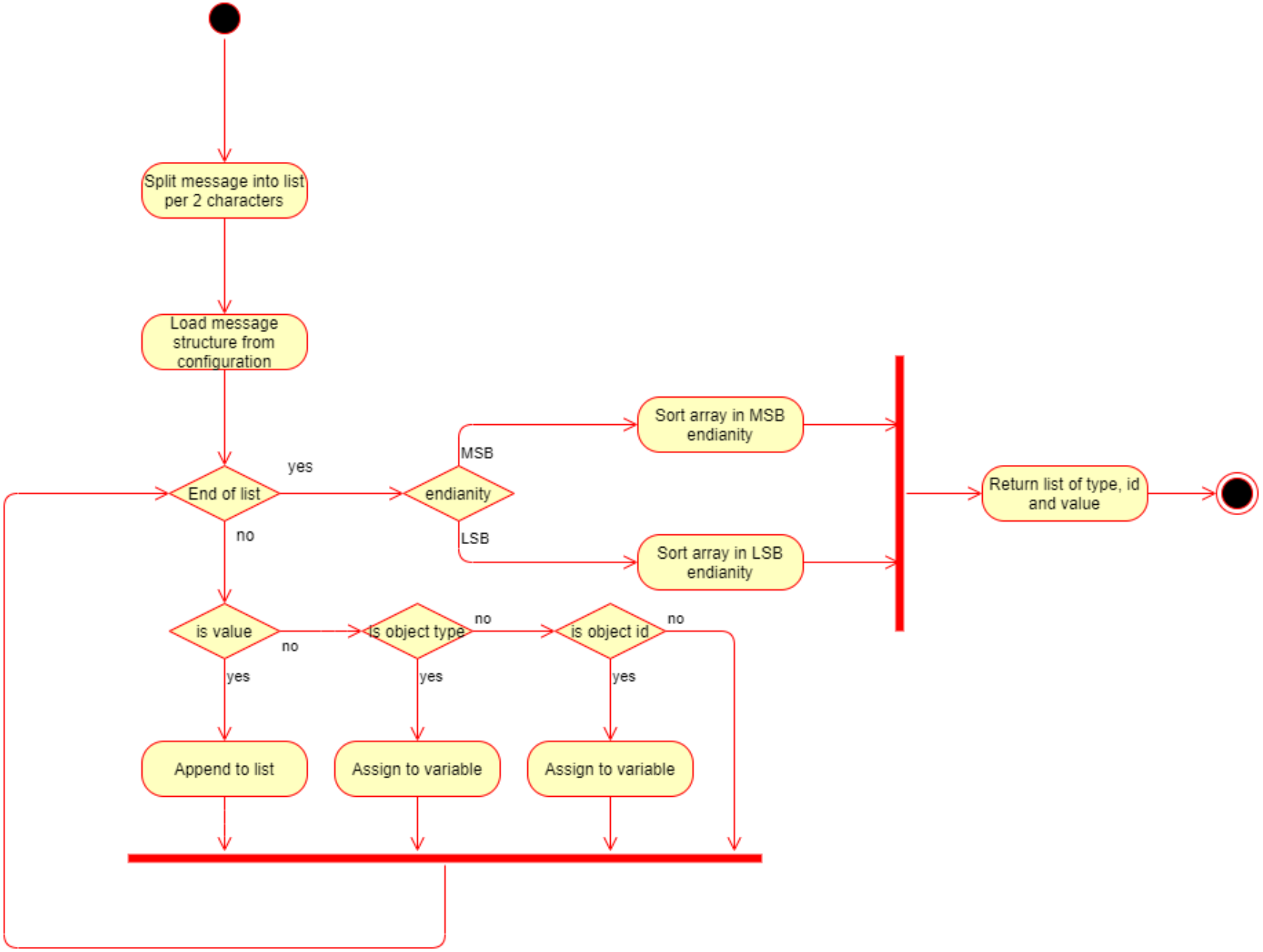
Obrázek 23: Sekvenční diagram metody initEventHandler



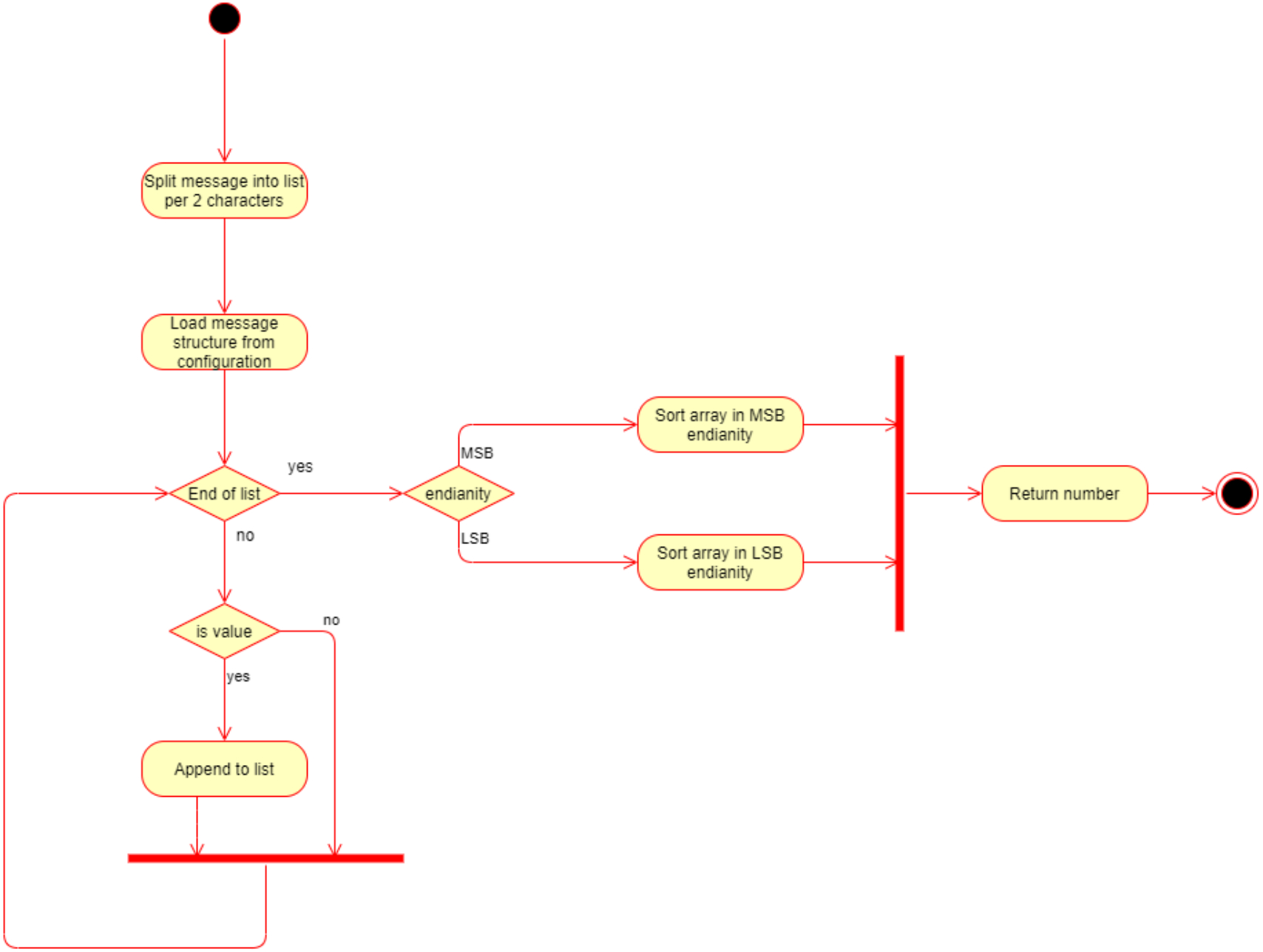
Obrázek 24: Diagram aktivit metody calculateChecksum ve třídě Checksum



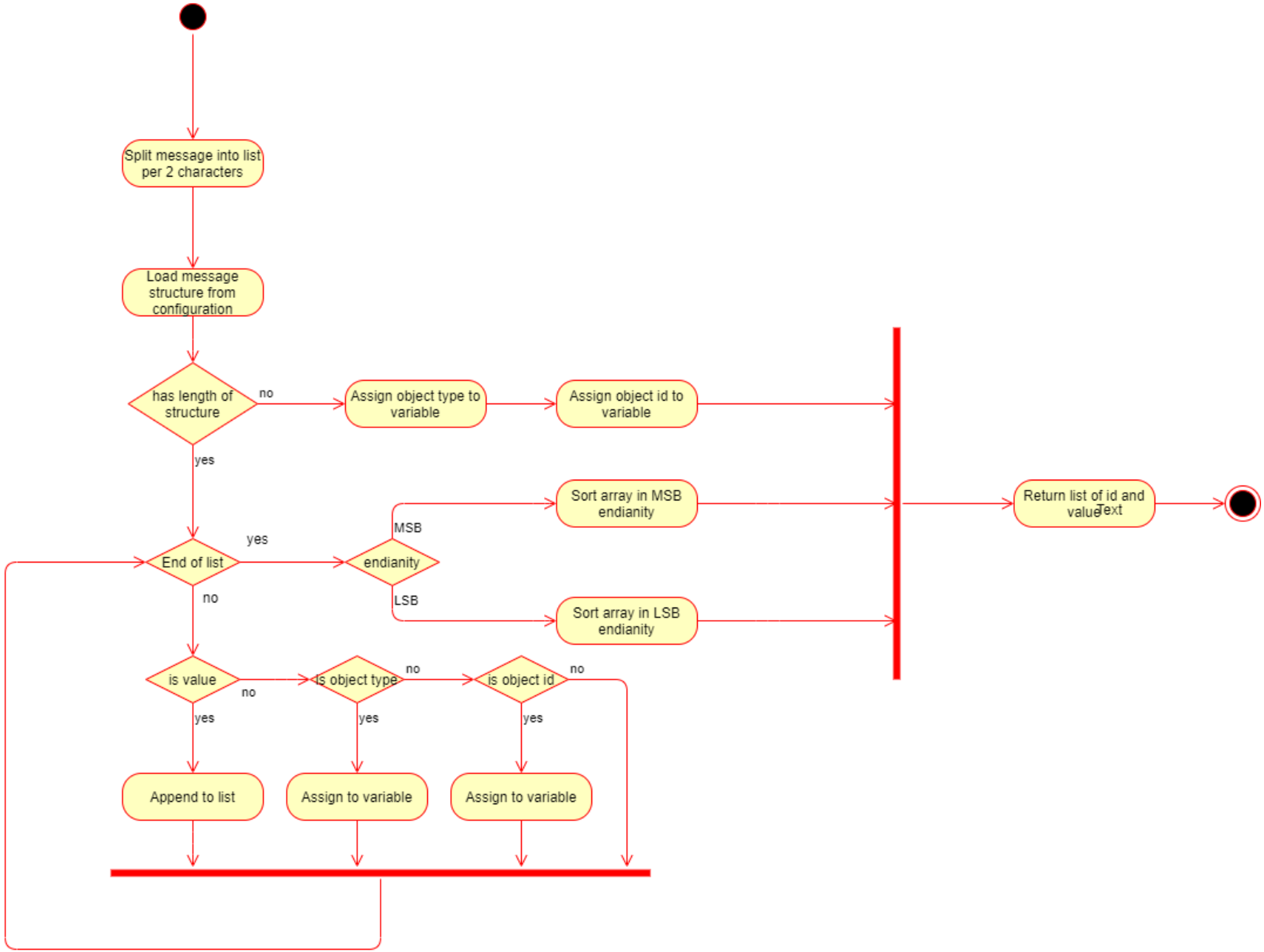
Obrázek 25: Diagram aktivit metody decodeToNumber ve třídě MessageDecoder4dsystems



Obrázek 26: Diagram aktivit metody decodeEvent ve třídě MessageDecoder4dsystems



Obrázek 27: Diagram aktivit metody decodeToNumber ve třídě MessageDecoderNextion



Obrázek 28: Diagram aktivit metody decodeEvent ve třídě MessageDecoderNextion

Tabulka 7: Jednotlivé displeje značky Nextion

Název	Velikost	Rozlišení	Typ dotyku	Rozhraní	Cena
NX3224T024	2,4 palců	240×320 pixelů	Rezistivní	UART	16 USD
NX3224T028	2,8 palců	240×320 pixelů	Rezistivní	UART	18 USD
NX4024T032	3,2 palců	240×400 pixelů	Rezistivní	UART	22 USD
NX4832T035	3,5 palců	320×480 pixelů	Rezistivní	UART	27 USD
NX4827T043	4,3 palců	272×480 pixelů	Rezistivní	UART	45 USD
NX8048T050	5,0 palců	800×480 pixelů	Rezistivní	UART	65 USD
NX8048T070	7,0 palců	800×480 pixelů	Rezistivní	UART	75 USD
NX3224K024	2,4 palců	240×320 pixelů	Rezistivní	UART	20 USD
NX3224K028	2,8 palců	240×320 pixelů	Rezistivní	UART	22 USD
NX4024K032	3,2 palců	240×400 pixelů	Kapacitní	UART	27 USD
NX4832K035	3,5 palců	320×480 pixelů	Rezistivní	UART	32 USD
NX4827K043	4,3 palců	272×480 pixelů	Rezistivní	UART	50 USD
NX8048K050	5,0 palců	800×480 pixelů	Rezistivní	UART	72 USD
NX8048K070	7,0 palců	800×480 pixelů	Rezistivní	UART	82 USD
NX8048K070-011R	7,0 palců	800×480 pixelů	Rezistivní	UART	88 USD
NX8048K070-011C	7,0 palců	800×480 pixelů	Kapacitní	UART	108 USD
NX8048P070-011R	7,0 palců	800×480 pixelů	Rezistivní	UART	78 USD
NX8048P070-011C	7,0 palců	800×480 pixelů	Kapacitní	UART	85 USD
NX1060P101-011R-I	10,1 palců	1024×600 pixelů	Rezistivní	UART	120 USD
NX1060P101-011C-I	10,1 palců	1024×600 pixelů	Kapacitní	UART	130 USD

Tabulka 8: Jednotlivé displeje značky MikroElektronika

Název	Velikost	Rozlišení	Typ dotyku	Rozhraní	Cena
HMI 3.5”	3,5 palců	320×240 pixelů	Žádný	CAN, SPI, I^{2}C, UART	74 USD
HMI 3.5” Res	3,5 palců	320×240 pixelů	Rezistivní	CAN, SPI, I^{2}C, UART	79 USD
HMI 3.5” Cap	3,5 palců	320×240 pixelů	Kapacitní	CAN, SPI, I^{2}C, UART	103 USD
HMI 4.3”	4,3 palců	480×272 pixelů	Žádný	CAN, SPI, I^{2}C, UART	79 USD
HMI 4.3” Res	4,3 palců	480×272 pixelů	Rezistivní	CAN, SPI, I^{2}C, UART	92 USD
HMI 4.3” Cap	4,3 palců	480×272 pixelů	Kapacitní	CAN, SPI, I^{2}C, UART	119 USD
HMI 5.0”	5,0 palců	800×480 pixelů	Žádný	CAN, SPI, I^{2}C, UART	108 USD
HMI 5.0” Res	5,0 palců	800×480 pixelů	Rezistivní	CAN, SPI, I^{2}C, UART	119 USD
HMI 5.0” Cap	5,0 palců	800×480 pixelů	Kapacitní	CAN, SPI, I^{2}C, UART	135 USD
HMI 7.0”	7,0 palců	800×480 pixelů	Žádný	CAN, SPI, I^{2}C, UART	121 USD
HMI 7.0” Res	7,0 palců	800×480 pixelů	Rezistivní	CAN, SPI, I^{2}C, UART	135 USD
HMI 7.0” Cap	7,0 palců	800×480 pixelů	Kapacitní	CAN, SPI, I^{2}C, UART	167 USD

Tabulka 9: Jednotlivé displeje značky Electronic Assembly

Název	Velikost	Rozlišení	Typ dotyku	Rozhraní	Cena
EA eDIP128W-6LW	2,8 palců	128×64 pixelů	Žádný	RS-232, I^{2}C, SPI	-
EA eDIP128W-6LWTP	2,8 palců	128×64 pixelů	Rezistivní	RS-232, I^{2}C, SPI	-
EA eDIP160W-7LW	3,2 palců	160×104 pixelů	Žádný	RS-232, I^{2}C, SPI	-
EA eDIP160W-7LWTP	3,2 palců	160×104 pixelů	Rezistivní	RS-232, I^{2}C, SPI	-
EA eDIP240J-7LW	4,2 palců	240×128 pixelů	Žádný	RS-232, I^{2}C, SPI	-
EA eDIP240J-7LWTP	4,2 palců	240×128 pixelů	Rezistivní	RS-232, I^{2}C, SPI	-
EA eDIP320J-8LW	5,7 palců	320×240 pixelů	Žádný	RS-232, I^{2}C, SPI	-
EA eDIP320J-8LWTP	5,7 palců	320×240 pixelů	Rezistivní	RS-232, I^{2}C, SPI	-

Tabulka 10: Jednotlivé displeje značky Riverdi

Název	Velikost	Rozlišení	Typ dotyku	Rozhraní	Cena
RiTFT-35	3,5 palců	240×320 pixelů	Žádný	SPI/QSPI	44 USD
RiTFT-35-CAP	3,5 palců	240×320 pixelů	Kapacitní	SPI/QSPI	65 USD
RiTFT-35-RES	3,5 palců	240×320 pixelů	Rezistivní	SPI/QSPI	47 USD
RiTFT-43	4,3 palců	480×272 pixelů	Žádný	SPI/QSPI	42 USD
RiTFT-43-CAP	4,3 palců	480×272 pixelů	Kapacitní	SPI/QSPI	66 USD
RiTFT-43-RES	4,3 palců	480×272 pixelů	Rezistivní	SPI/QSPI	46 USD
RiTFT-50	5,0 palců	800×480 pixelů	Žádný	SPI/QSPI	58 USD
RiTFT-50-CAP	5,0 palců	800×480 pixelů	Kapacitní	SPI/QSPI	86 USD
RiTFT-50-RES	5,0 palců	800×480 pixelů	Rezistivní	SPI/QSPI	63 USD
RiTFT-70	7,0 palců	800×480 pixelů	Žádný	SPI/QSPI	65 USD
RiTFT-70-CAP	7,0 palců	800×480 pixelů	Kapacitní	SPI/QSPI	104 USD
RiTFT-70-RES	7,0 palců	800×480 pixelů	Rezistivní	SPI/QSPI	78 USD

Tabulka 11: 4D Systems zobrazovací prvky

Název	ID objektu	Typ
Dipswitch	0x00	Vstupní
Knob	0x01	Vstupní
Rockerswitch	0x02	Vstupní
Rotaryswitch	0x03	Vstupní
Slider	0x04	Vstupní
Trackbar	0x05	Vstupní
Winbutton	0x06	Vstupní
Angularmeter	0x07	Výstupní
Coolgauge	0x08	Výstupní
Customdigits	0x09	Výstupní
Form	0x0a	Výstupní
Gauge	0x0b	Výstupní
Image	0x0c	-
Keyboard	0x0d	Výstupní
Led	0x0e	Výstupní
Leddigits	0x0f	Výstupní
Meter	0x10	Výstupní
Strings	0x11	Výstupní
Thermometer	0x12	Výstupní
Userled	0x13	Výstupní
Video	0x14	Výstupní
Statictext	0x15	-
Sound	0x16	Výstupní
Timer	0x17	Výstupní
Spectrum	0x18	Výstupní
Scope	0x19	Výstupní
Tank	0x1a	Výstupní
UserImages	0x1b	Výstupní
PinOutput	0x1c	Výstupní
PinInput	0x1d	Vstupní
4Dbutton	0x1e	Vstupní
AniButton	0x1f	Vstupní
ColorPicker	0x20	Vstupní
UserButton	0x21	Vstupní

Tabulka 12: Nextion zobrazovací prvky

Název	Typ
Text	Výstupní
Scrolling text	Výstupní
Number	Výstupní
Xfloat	Výstupní
Button	Vstupní
Progress bar	Výstupní
Picture	-
Crop	-
Hotspot	-
Gauge	Výstupní
Waveform	Výstupní
Slider	Vstupní
Timer	-
Variable	-
Dual-state Button	Vstupní
Checkbox	Vstupní
Radio	Vstupní
QRcode	Výstupní

Tabulka 13: Surenoo zobrazovací prvky

Název	Typ
Square	-
Line	-
Label	Výstupní
Number	Výstupní
Form	Vstupní
Image	Výstupní
Button	Vstupní
Box	-
Edit	Vstupní
Checkbox	Vstupní
Slider	Vstupní
ProgressBar	Výstupní
CircleGauge	Výstupní
BarGauge	Výstupní
WaterGauge	Výstupní
Thermometer	Výstupní
Waveform	Výstupní
Battery	Výstupní

Tabulka 14: Surenoo příkazy

Funkce	Příkaz	Délka	Příklad
zápis Text	0x74 <cid> <pid> <text>	x + 3 bajty	0x74, 0x01, 0x01, 0x61, 0x62, 0x63
zápis Number	0x6E <cid> <pid> <val> <val>	5 bajtů	0x6E, 0x01, 0x01, 0x31, 0x30
zápis Progress bar	0x6F <cid> <pid> <val>	4 bajty	0x6F, 0x01, 0x01, 0x31
zápis Checkbox	0x63 <cid> <pid> <val>	4 bajty	0x6F, 0x01, 0x01, 0x55
zápis CircleGauge	0x7A <cid> <pid> <val> <val>	5 bajtů	0x7A, 0x01, 0x01, 0x31, 0x30
zápis BarGauge	0x61 <cid> <pid> <val> <val>	5 bajtů	0x61, 0x01, 0x01, 0x31, 0x30
zápis WaterGauge	0x77 <cid> <pid> <val> <val>	5 bajtů	0x77, 0x01, 0x01, 0x31, 0x30
zápis Thermometer	0x6D <cid> <pid> <val> <val>	5 bajtů	0x6D, 0x01, 0x01, 0x31, 0x30
zápis Battery	0x79 <cid> <pid> <val> <val>	5 bajtů	0x79, 0x01, 0x01, 0x31, 0x30
zápis Slider	0x5F <cid> <pid> <val>	4 bajty	0x5F, 0x01, 0x01, 0x10
zápis Checkbox	0x64 <cid> <pid>	3 bajty	0x64, 0x01, 0x01
čtení Slider	0x68 <cid> <pid>	6 bajtů	0x68, 0x01, 0x01
vrácení čtení Checkbox	0x64 <cid> <pid> <sta> <type> <val>	6 bajtů	0x64, 0x01, 0x01, 0x6F, 0xFF, 0x55
vrácení čtení Slider	0x68 <cid> <pid> <sta> <type> <val>	6 bajtů	0x68, 0x01, 0x01, 0x6F, 0xFF, 0x10
vrácení události Button	0x62 <cid> <pid> <sta> <type> <val>	6 bajtů	0x62, 0x01, 0x01, 0x55, 0x36, 0x01
vrácení události Checkbox	0x60 <cid> <pid> <sta> <type> <val>	6 bajtů	0x60, 0x01, 0x01, 0x6F, 0xFF, 0x55
vrácení události Slider	0x67 <cid> <pid> <sta> <type> <val>	6 bajtů	0x67, 0x01, 0x01, 0x55, 0xFF, 0x10

Tabulka 15: 4D Systems uLCD-32PTU hardwarové specifikace

Parametr	Hodnota
Velikost displeje	3,2 palců ulopříčně
Rozlišení	240 × 320 pixelů
Velikost úložiště	14 KB
Operační paměť	14 KB
Komunikační rozhraní	UART
Slot pro microSD kartu	Ano
Velikost displeje	54,34 × 77,70 × 3,70mm
Velikost vizuální plochy	48,60 × 64,80mm
Velikost pixelu	0,2025 × 0,2025mm
Jas	200cd/m2
Poměr kontrastu	350:1
Viditelný úhel nad centrem	60 stupňů
Viditelný úhel pod centrem	60 stupňů
Viditelný úhel nalevo od centra	70 stupňů
Viditelný úhel napravo od centra	70 stupňů
Viditelný směr	12 hodin
Podsvícení	6 × LED

Tabulka 16: Nextion nx3224k032 hardwarové specifikace

Parametr	Hodnota
Počet barev	65536 barev
Velikost	74,4 × 2,72 × 5,8mm
Aktivní plocha	60,26 × 42,72mm
Vizuální plocha	48,96 × 36,72mm
Operační paměť	3584 bajtů
Velikost úložiště	16 MB
Komunikační rozhraní	UART
Slot pro microSD kartu	Ano
Rozlišení	320 × 240 pixelů
Velikost displeje	2,4 palců
Typ dotyku	Rezistivní
Životnost dotyků	>1 milión
Podsvícení	LED
Životnost podsvícení	>30000 hodin
Jas	180 nit
Váha	25,8g

Tabulka 17: Surenoo H28A-I hardwarové specifikace

Parametr	Hodnota
Rozlišení	320 × 240 pixelů
Velikost displeje	2,8 palců
Velikost úložiště	48 MB
Komunikační rozhraní	I^2C
Velikost obrazovky	61 × 89mm
Velikost zobrazovací plochy	41 × 57mm
Velikost pixelu	0,18 × 0,18mm
Počet barev	65536 barev
Vizuální úhel	12 hodin
Podsvícení	Bílá LED
Požadované napětí	3,3 5 V

B Zdrojové kódy knihovny

```
"""
The smart display library interface
"""

import json
import os.path

from .connectionHandlers.ConnectionHandlerFactory import
    ConnectionHandlerFactory
from .messageDecoders.MessageDecoderFactory import MessageDecoderFactory
from .messageHandlers.MessageHandlerFactory import MessageHandlerFactory

class DisplayHandler:
    """
    DisplayHandler class is used as library's interface
    """
    def __init__(self, fileName):
        """
        Constructs new object of DisplayHandler.

        :param fileName: The name of the configuration file
        :return: None
        """
        filePath = os.path.dirname(__file__) + '/../displayConfigurations/' +
            fileName + '.json'
        with open(filePath,"r") as myfile:
            data=myfile.read()
        self.displayConf = json.loads(data)

        self.msgHandlerFactory = MessageHandlerFactory()
        self.msgDecoderFactory = MessageDecoderFactory()
        self.conHandlerFactory = ConnectionHandlerFactory()

        self.msgHandler = self.msgHandlerFactory.getMessageHandler(self.
            displayConf["name"])
        self.msgDecoder = self.msgDecoderFactory.getMessageDecoder(self.
            displayConf["name"], self.displayConf)
```

```
        self.conHandler = self.conHandlerFactory.getConnectionHandler(self.
            displayConf["communicationType"], self.displayConf)


    def getIdsByItem(self, item):
        """
        getIdsByItem is used for getting all identificators inside
            configuration file

        :param item: The name of the item from configuration file
        :return: Array of ids by specified item
        """
        return self.displayConf["elements"][item]["indexes"]

    def getItemsWithId(self):
        """
        getItemsWithId is used for getting all item's name
        inside configuration file, which has atleast one id inside

        :return: Array of item's names
        """
        result = []
        for item in self.displayConf["elements"]:
            if len(self.displayConf["elements"][item]["indexes"]) > 0:
                result.append(item)
        return result

    def setValue(self, item, id, value):
        """
        setValue is used for sending message to specified item on the smart
            display

        :param item: The name of the item from configuration file
        :param id: The index of the specified item from configuration file
        :param value: The value which should be sent to smart display
        :return: None
        """

        if type(value) is int:
```

```
            message = self.msgHandler.structureSetValMessage(self.displayConf,
                item, id, value)
        elif type(value) is str:
            message = self.msgHandler.structureSetTextMessage(self.displayConf,
                item, id, value)
        else:
            return "Error"

        returnLen = len(self.displayConf["returnSetStructure"])
        self.conHandler.sendMessage(message, returnLen)

    def getValue(self, item, id):
        """
        getValue is used for getting value of specified item on the smart
            display

        :param item: The name of the item from configuration file
        :param id: The index of the specified item from configuration file
        :return: Number of specified item
        """
        message = self.msgHandler.structureGetValMessage(self.displayConf, item
            , id)

        returnLen = len(self.displayConf["returnGetStructure"])
        response = self.conHandler.sendMessage(message, returnLen)

        decodedMessage = self.msgDecoder.decodeToNumber(response)

        return decodedMessage

    def initEventHandler(self):
        """
        initEventHandler is used for checking if user input has happend.

        :return: Array which consists of specified item and its value
        """
        returnLen = len(self.displayConf["returnEventStructure"])
        response = self.conHandler.checkEvent(returnLen)
```

```
decodedMessage = ""
if (response!=""):
    decodedMessage = self.msgDecoder.decodeEvent(response)

return decodedMessage
```

Výpis 6: Třída DisplayHandler

C Konfigurační soubory knihovny

```
{
    "name":"4dsystems",
    "communicationType":"UART",
    "baudrate":9600,
    "communicationInterface":"/dev/serial0",
    "endian":"MSB",

    "commands":{
        "read":"00",
        "write":"01",
        "writeStr":"02",
        "writeStru":"03",
        "return":"05",
        "ACK":"06",
        "NACK":"15"
    },

    "getValStructure": [
        "read","objectType","objectId","checksum"
    ],
    "writeValStructure": [
        "write","objectType","objectId","value","value","checksum"
    ],
    "writeTextStructure": [
        "writeStr","objectId","length","value","checksum"
    ],
    "returnGetStructure": [
        "return","objectType","objectId","value","value","checksum"
    ],
    "returnSetStructure": [
        "ACK"
    ],
    "returnEventStructure": [
        "event","objectType","objectId","value","value","checksum"
    ],

    "elements":{
```

```
"dipswitch":{
    "address":"00",
    "value":"number",
    "output":true,
    "indexes":[]
},
"knob":{
    "address":"01",
    "value":"number",
    "output":true,
    "indexes":[]
},
"rockerswitch":{
    "address":"02",
    "value":"number",
    "output":true,
    "indexes":[]
},
"rotaryswitch":{
    "address":"03",
    "value":"number",
    "output":true,
    "indexes":[]
},
"slider":{
    "address":"04",
    "value":"number",
    "output":true,
    "indexes":[]
},
"trackbar":{
    "address":"05",
    "value":"number",
    "output":true,
    "indexes":[]
},
"winbutton":{
    "address":"06",
    "value":"number",
```

```
        "output":true,
        "indexes":[]
    },
    "angularmeter":{
        "address":"07",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "coolgauge":{
        "address":"08",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "customdigits":{
        "address":"09",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "form":{
        "address":"0a",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "gauge":{
        "address":"0b",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "led":{
        "address":"0e",
        "value":"number",
        "output":true,
        "indexes":[]
    },
```

```
"leddigits":{
    "address":"0f",
    "value":"number",
    "output":true,
    "indexes":[]
},
"meter":{
    "address":"10",
    "value":"number",
    "output":true,
    "indexes":[]
},
"strings":{
    "address":"11",
    "value":"text",
    "output":true,
    "indexes":[]
},
"thermometer":{
    "address":"12",
    "value":"number",
    "output":true,
    "indexes":[]
},
"userled":{
    "address":"13",
    "value":"number",
    "output":true,
    "indexes":[]
},
"video":{
    "address":"14",
    "value":"number",
    "output":true,
    "indexes":[]
},
"statictext":{
    "address":"15",
    "value":"number",
```

```
        "output":true,
        "indexes":[]
    },
    "sound":{
        "address":"16",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "timer":{
        "address":"17",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "spectrum":{
        "address":"18",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "scope":{
        "address":"19",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "tank":{
        "address":"1a",
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "userImages":{
        "address":"1b",
        "value":"number",
        "output":true,
        "indexes":[]
    },
```

```
        "pinOutput":{
            "address":"1c",
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "pinInput":{
            "address":"1d",
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "4dButton":{
            "address":"1e",
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "aniButton":{
            "address":"1f",
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "colorPicker":{
            "address":"20",
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "userButton":{
            "address":"21",
            "value":"number",
            "output":true,
            "indexes":[]
        }
    }
}
```

Výpis 7: Konfigurační soubor pro displeje 4D Systems

```
{
    "name":"nextion",
    "communicationType":"UART",
    "baudrate":9600,
    "communicationInterface":"/dev/serial0",
    "endian":"LSB",

    "commands":{
        "dot":".",
        "space":" ",
        "read":"get",
        "write":"=",
        "number":"71",
        "string":"70",
        "end":"ÿ",
        "id":"z0",
        "val":"val",
        "text":"txt",
        "value":"120"
    },

    "getValStructure": [
        "read","space","objectId","dot","val","end","end","end"
    ],
    "writeValStructure": [
        "objectId","dot","val","write","value","end","end","end"
    ],
    "writeTextStructure": [
        "objectId","dot","text","write","value","end","end","end"
    ],
    "returnGetStructure": [
        "type","value","value","value","value","end","end","end"
    ],
    "returnSetStructure": [

    ],
```

```
"returnEventStructure": [
    "objectType","objectId","value","value","value","value"
],

"elements":{
    "text":{
        "value":"text",
        "output":true,
        "indexes":["t0"]
    },
    "scollingText":{
        "value":"text",
        "output":true,
        "indexes":[]
    },
    "number":{
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "xFloat":{
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "button":{
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "progressBar":{
        "value":"number",
        "output":true,
        "indexes":[]
    },
    "picture":{
        "value":"number",
        "output":true,
        "indexes":[]
```

```
},
"crop":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"hotspot":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"gauge":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"waveform":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"slider":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"timer":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"variable":{
    "value":"number",
    "output":true,
    "indexes":[]
},
"dualStateButton":{
    "value":"number",
    "output":true,
```

```
            "indexes":[]
        },
        "checkbox":{
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "radio":{
            "value":"number",
            "output":true,
            "indexes":[]
        },
        "qrCode":{
            "value":"number",
            "output":true,
            "indexes":[]
        }
    }
}
```

Výpis 8: Konfigurační soubor pro displeje Nextion